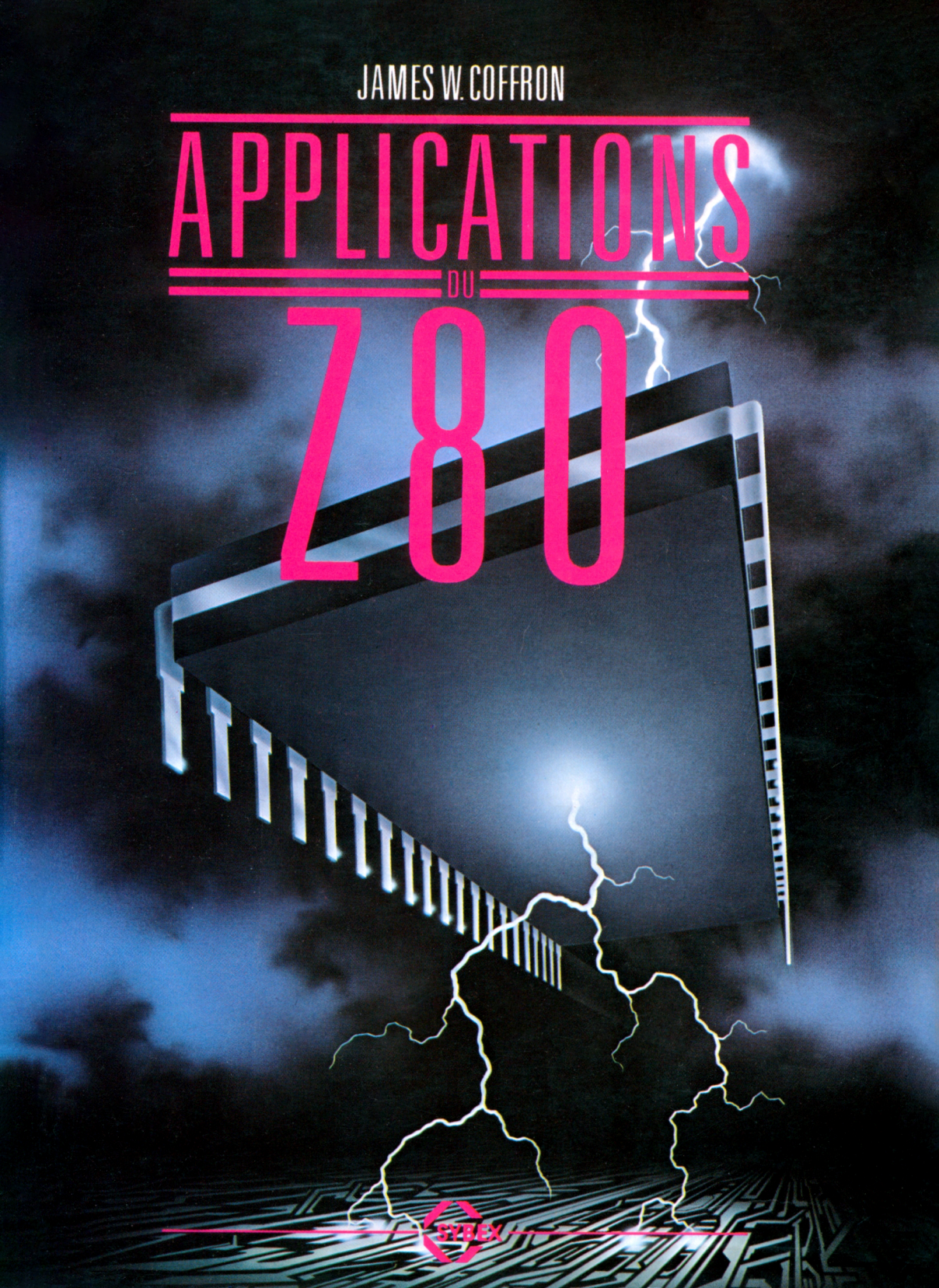
JAMES W. COFFRON

APPLICATIONS DU Z80



SYBEX

JAMES W. COFFRON

APPLICATIONS DU Z80

Traduction française : **Dominique de Pardieu**

Couverture de **Jean-François Pénichoux**

Z80 est une marque déposée de ZILOG Inc.
MOSTEK est une marque déposée de Mostek Inc.
Intel est une marque déposée de Intel Inc.

Tous les efforts ont été faits pour fournir dans ce livre une information complète et exacte. Néanmoins, SYBEX n'assume de responsabilité ni pour son utilisation ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

Copyright version originale © 1983 SYBEX Inc.
version française © 1984 SYBEX

Tous droits réservés. Toute reproduction même partielle, par quelque procédé que ce soit, est interdite sans autorisation préalable. Une copie par xérographie, photographie, film, bande magnétique ou autre, constitue une contrefaçon passible des peines prévues par la loi sur la protection des droits d'auteur.

ISBN 2-902414-74-9
(Edition originale : ISBN 0-89588-094-6 SYBEX Inc., Berkeley)

JAMES W. COFFRON

APPLICATIONS DU Z80

SYBEX

Paris • Berkeley • Düsseldorf

TABLE DES MATIERES

Introduction

Chapitre 1 UTILISATION DU Z80 AVEC DES ROM

Introduction 1
1-1 *Qu'est-ce qu'une ROM ?* 1
1-2 *Principales caractéristiques des ROM* 3
1-3 *Déroulement des opérations lors d'une lecture des données dans une ROM* 5
1-4 *Connexion des bus du Z80* 6
1-5 *Plan d'adressage* 8
1-6 *Production des signaux de sélection de boîtier pour ROM* 10
1-7 *Production des signaux de lecture de mémoire* 11
1-8 *Connexion des lignes de sélection de boîtier (chip select)* 13
1-9 *Variante* 14
1-10 *Addition de plusieurs ROM* 15
1-11 *Etablissement d'un plan d'adressage pour ROM plus importantes* 16
1-12 *Amplificateurs de bus d'adresses* 18
1-13 *Amplification des données en mémoire* 21
1-14 *Trois exemples de système à ROM complets* 25
Résumé 25

Chapitre 2 UTILISATION DE RAM STATIQUES AVEC LE Z80

Introduction 27

2-1 *Aperçu sur la manière de communiquer avec des RAM statiques* 27

2-2 *Suite des opérations pendant une lecture de RAM* 29

2-3 *Suite des opérations pour une écriture en RAM* 31

2-4 *Mémoire réelle* 32

2-5 *Suite des opérations d'écriture de données dans une 2114* 36

2-6 *Suite des opérations de lecture d'une RAM 2114* 37

2-7 *Connexion des lignes d'adresse au Z80* 37

2-8 *Connexion des lignes de données - sans amplification* 41

2-9 *Production des commandes de lecture et d'écriture* 42

2-10 *Utilisation de lignes de données amplifiées avec des RAM statiques* 45

2-11 *RAM statique complète 4K × 8 bits* 45

2-12 *Autre RAM statique : la 6116* 45

Résumé 52

Chapitre 3 ENTREES ET SORTIES DU Z80

Introduction 55

3-1 *Aperçu sur les entrées et sorties du Z80* 55

3-2 *Adresses de port* 56

3-3 *Production des commandes $\overline{IOW}$ et $\overline{IOR}$* 59

3-4 *Production du signal de lecture de port* 62

3-5 *Schéma complet d'un port d'entrée-sortie* 64

3-6 *Déroulement des opérations pendant une écriture en sortie* 64

3-7 *Opération de lecture dans un port* 66

3-8 *Récapitulation des opérations* 67

3-9 *Séquence d'écriture en sortie* 67

3-10 *Séquence de lecture en entrée* 68

Résumé 68

Chapitre 4 UTILISATION DE RAM DYNAMIQUES AVEC LE Z80

Introduction 71

4-1 *Aperçu de la 4116* 71

4-2 *Multiplexage des lignes d'adresses* 76

4-3 *Schéma du système à RAM dynamique 16K × 8 bits* 78

4-4 *Production des signaux $\overline{RAS}$, $\overline{CAS}$ et MUX* 80

4-5 *Entrées des données dans la RAM dynamique* 81
4-6 *Ecriture de données dans une RAM dynamique* 83
4-7 *Sortie des données des RAM dynamiques* 86
4-8 *Rafraîchissement des RAM dynamiques* 88
4-9 *Schéma complet d'une RAM dynamique 16K × 8 bits* 91
Résumé 91

Chapitre 5 INTERRUPTIONS DU Z80

Introduction 93
5-1 *Qu'est-ce qu'une interruption ?* 93
5-2 *D'où les demandes d'interruptions proviennent-elles ?* 94
5-3 *Interruptions non masquables* 95
5-4 *Effacement de la demande $\overline{NMI}$* 98
5-5 *Fin de la routine de gestion $\overline{NMI}$* 98
5-6 *Exemple de fonctionnement avec $\overline{NMI}$* 98
5-7 *Résumé des opérations $\overline{NMI}$* 101
5-8 *Entrée $\overline{INT}$* 101
5-9 *Interruption en mode 1* 102
5-10 *Interruption en mode 0* 105
5-11 *Interruption en mode 2* 109
5-12 *Interruptions demandées par plusieurs dispositifs* 114
5-13 *Scrutation* 115
5-14 *Interruptions prioritaires* 115
5-15 *Priorité à chaînage* 118
Résumé 119

Chapitre 6 UTILISATION DU PIO 8255 AVEC LE Z80

Introduction 121
6-1 *Aperçu général du 8255* 121
6-2 *Brochage du 8255* 123
6-3 *Liaison du 8255 avec le CPU Z80* 125
6-4 *Registres de lecture et d'écriture du 8255* 127
6-5 *Mode 0 - Registre d'entrée-sortie fondamental* 129
6-6 *Exemple de mode 0* 131
6-7 *Fonctionnement en mode 1 du 8255* 136
6-8 *Fonctionnement en mode 2 du 8255* 143
Résumé 145

Chapitre 7 UTILISATION DU TEMPORISATEUR PROGRAMMABLE 8253 AVEC LE Z80

Introduction 147
7-1 *Schéma du temporisateur programmable 8253* 147

7-2 *Les trois lignes des compteurs : horloge, déclenchement et sortie* 148
7-3 *Registres internes du 8253* 149
7-4 *Connexion du 8253 au Z80* 151
7-5 *Programmation du 8253 (format du mot de commande)* 154
7-6 *Exemple de mode 0 : Interruption à la fin du comptage* 158
7-7 *Mode 1 : monostable programmable* 161
7-8 *Mode 2 : générateur d'impulsions* 161
7-9 *Mode 3 : générateur de signaux carrés* 163
7-10 *Mode 4 : échantillonnage déclenché par logiciel* 165
7-11 *Exemple* 165
7-12 *Mode 5 : échantillonnage déclenché par circuit* 166
7-13 *Utilisation des broches d'entrée de déclenchement* 166
Résumé 168

Chapitre 8 UTILISATION DU PIO DU Z80

Introduction 171
8-1 *Schéma du PIO* 171
8-2 *Brochage du PIO du Z80* 172
8-3 *Connexion du PIO du Z80 au Z80* 177
8-4 *Mise à l'état initial du PIO* 178
8-5 *Programmation du PIO en mode 0 (port de sortie)* 178
8-6 *Mode de programmation 1* 179
8-7 *Etablissement du mot de commande d'interruption* 184
8-8 *Examen des temps en mode 0 et 1* 187
8-9 *Utilisation du PIO en mode bidirectionnel (mode 2)* 188
8-10 *Utilisation du PIO en mode 3* 191
8-11 *Autorisation et inhibition d'interruption* 195
8-12 *Priorité d'interruption du PIO* 195
Résumé 197

Chapitre 9 UTILISATION DU CTC DU Z80

Introduction 199
9-1 *Schéma du CTC* 199
9-2 *Examen plus détaillé du bloc de canaux* 200
9-3 *Brochage du CTC du Z80* 201
9-4 *Définitions des signaux du CTC* 201
9-5 *Connexion du CTC au Z80* 204
9-6 *Examen du mode compteur du CTC* 207
9-7 *Programmation du registre de commande du canal* 209
9-8 *Programmation du registre de constante de temps* 211
9-9 *Programmation du vecteur d'interruption* 211

9-10 *Programmation du CTC pour fonctionnement en compteur* 212
9-11 *Exemple de fonctionnement du CTC en mode temporisateur* 216
Résumé 218

Chapitre 10 INTRODUCTION AUX COMMUNICATIONS SERIE

Introduction 221
10-1 *Qu'est-ce qu'une communication série* 221
10-2 *Chronologie série* 222
10-3 *Conversion de données en parallèle en données série* 224
10-4 *Bit de départ* 225
10-5 *Bit de parité* 225
10-6 *Bit d'arrêt* 227
10-7 *Récapitulation des principales notions sur la communication série* 227
10-8 *Examen du 8251* 229
10-9 *Brochage du 8251* 232
10-10 *Connexion du 8251 aux bus du Z80* 234
10-11 *Connexions série* 236
10-12 *Programmation du 8251* 237
10-13 *Erreur de cadrage* 245
10-14 *Erreur de surcharge* 245
10-15 *Programme d'application simple pour 8251* 245
10-16 *Développement d'une application pour 8251* 245
Résumé 249

Chapitre 11 UTILISATION DU Z80-SIO

Introduction 253
11-1 *Schéma du SIO* 253
11-2 *Brochage du SIO* 254
11-3 *Connexion du SIO aux bus du Z80* 259
11-4 *Connexion du SIO aux lignes de transmission série* 262
11-5 *Registres du SIO* 262
11-6 *Séquence générale d'initialisation du SIO* 263
11-7 *Réception de données série* 265
11-8 *Transmission de caractères en mode scruté* 267
11-9 *Interruptions avec le SIO* 271
11-10 *Initialisation du SIO pour les interruptions* 273
11-11 *Après l'initialisation* 273
Résumé 277

Chapitre 12 ESSAIS DU Z80 PAR STIMULATION STATIQUE

Introduction 279
12-1 *Aperçu des essais par stimulation statique* 280
12-2 *Circuits du testeur statique à stimulation* 283
12-3 *Lignes de sortie d'adresses et de données du SST* 285
12-4 *$\overline{MI}$, $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, $\overline{WR}$, $\overline{RFSH}$, $\overline{HALT}$, $\overline{BUSAK}$* 286
12-5 *Affichage par LED pour le bus de données* 288
Résumé 289

Annexe : Description des registres internes du Z80-SIO

Index

INTRODUCTION

Vous est-il déjà arrivé d'imaginer une application nouvelle pour un système commandé par microprocesseur, pour voir ensuite votre idée s'évanouir sous le flot des détails techniques trouvés dans les feuilles de spécifications ? Si oui, vous n'êtes pas le seul. Plusieurs remarquables applications pour systèmes commandés par microprocesseur ont connu ce destin. L'idée était saine, mais son auteur ne connaissait pas suffisamment les détails du microprocesseur pour réaliser complètement le projet.

Ce livre a été écrit pour aider les utilisateurs de système à Z80 à surmonter cette difficulté. Il a été conçu pour faire comprendre le microprocesseur Z80 à ses utilisateurs. D'une façon générale, il explique en détail comment on peut connecter le microprocesseur Z80 aux circuits pour former un système. D'une façon plus particulière, il explique les ROM, les RAM statiques, les RAM dynamiques et les circuits d'entrée-sortie. En outre, les derniers chapitres étudient certains dispositifs spéciaux d'entrée-sortie. Certains de ces dispositifs, comme le SIO (entrée et sortie série) et le PIO (entrée et sortie pour périphériques) ont été fabriqués pour être utilisés avec le Z80, contrairement à d'autres comme le temporisateur 8253 et le boîtier 8255.

Ecrit pour le débutant comme pour le programmeur expérimenté, cet ouvrage offre un texte instructif, des schémas clairs et précis et des exemples complets. Il a été rédigé pour donner aux utilisateurs des renseignements leur permettant de réaliser et d'exécuter leurs propres applications en mettant en oeuvre leurs idées. Bien que ce livre ne présente pas la totalité des applications du microprocesseur Z80 - c'est à vous de trouver des applications individuelles - il fournit les renseignements permettant de réaliser n'importe quelle nouvelle application.

Contenu

Le Chapitre 1, *Utilisation du Z80 avec des ROM*, débute par une explication des différents types de mémoires à lecture seule. Il montre ensuite comment des dispositifs ROM et EPROM ordinaires peuvent être interfacés au CPU Z80.

Le Chapitre 2, *Utilisation de RAM statiques avec le Z80,* montre comment on peut utiliser une mémoire à accès aléatoire statique dans une application comportant un Z80. Il analyse les boîtiers RAM aussi bien à entrée-sortie communes qu'à entrée-sortie séparées et présente les schémas complets de deux systèmes à RAM statiques.

Le Chapitre 3, *Entrées et sorties du Z80,* explique comment le Z80 communique électriquement avec les boîtiers d'entrée-sortie. Des ports d'entrée et de sortie standard sont représentés et analysés en détail.

Le Chapitre 4, *Utilisation de RAM dynamiques avec le Z80,* montre comment on peut employer des systèmes à RAM dynamiques avec un Z80. Ce chapitre commence par un examen d'une RAM dynamique type. Il explique ensuite comment on peut interfacer une RAM au Z80, grâce à la structure interne de ce dernier.

Le Chapitre 5, *Interruptions du Z80*, analyse ce point important que constituent les interruptions. Les trois modes d'interruption sont examinés et des exemples de chaque type sont donnés.

Le Chapitre 6, *Utilisation du PIO 8255 avec le Z80*, examine l'emploi du boîtier d'entrée-sortie 8255 pour périphériques, dispositif interfacé aux bus du Z80. Chaque mode de fonctionnement du 8255 est analysé et de nombreux exemples de programmation sont introduits.

Le Chapitre 7, *Utilisation du temporisateur programmable 8253 avec le Z80*, montre comment le boîtier temporisateur programmable 8253 peut être employé avec le Z80. On explique comment on peut interfacer ce boîtier au Z80. De plus, plusieurs exemples de programmation générale sont présentés.

Le Chapitre 8, *Utilisation du PIO du Z80,* continue l'étude du boîtier pour périphériques en s'attachant au Z80-PIO. Il donne un schéma de ce composant et explique les modes de fonctionnement et les registres. En outre, il propose de nombreux exemples de programmation avec PIO.

Le Chapitre 9, *Utilisation du CTC du Z80,* explique le boîtier temporisateur-compteur et montre comment l'interfacer au Z80. De plus, de nombreux exemples de programmation avec CTC sont analysés.

Le Chapitre 10, *Introduction aux communications série,* explique le sujet des communications en série. Des notions, telles que le nombre de bauds, le bit de départ, le bit d'arrêt et les niveaux de marquage, sont définies. Comme exemple pratique, on montre comment interfacer et employer l'USART 8251 avec le Z80.

Le Chapitre 11, *Utilisation du Z80-SIO*, continue les communications série en expliquant le boîtier Z80-SIO. Il donne plusieurs exemples pratiques d'utilisation et de programmation du SIO.

Le Chapitre 12, *Essais du Z80 par stimulation statique,* conclut le texte par une analyse de l'essai par stimulation statique (SST) du Z80. Il montre comment on peut employer cette méthode pour mettre au point un système sans logiciel. De plus, si l'on ajoute une nouvelle interface à un système à Z80 existant, on peut employer cette technique pour vérifier la nouvelle interface d'une façon rapide et facile.

Enfin, l'Annexe, *Description des registres internes du Z80-SIO*, décrit le fonctionnement de chaque registre interne du SIO.

Comme on peut le voir, ce livre donne des renseignements sur tous les sujets importants de façon à permettre la compréhension du microprocesseur Z80. Avec les connaissances qu'il permet d'acquérir, on doit pouvoir mettre en oeuvre n'importe quelle idée ou application de système commandé par microprocesseur. Lorsque vous commencerez à réaliser vos propres idées et applications, vous trouverez que l'emploi et l'application du microprocesseur Z80 ne sont pas seulement faciles, mais peuvent aussi être une source de beaucoup de plaisir. Laissez donc libre cours à votre imagination, lisez et apprenez comment mettre le Z80 au travail pour vous.

UTILISATION
DU Z80 AVEC DES ROM

CHAPITRE 1

INTRODUCTION

Nous allons débuter notre étude du microprocesseur Z80 en apprenant comment le relier à une mémoire à lecture seule (ROM). Nous commencerons par une présentation générale du fonctionnement de ces mémoires. Nous examinerons ensuite le point important qu'est l'utilisation des ROM avec le Z80. Vers la fin de ce chapitre, vous devriez comprendre totalement comment une ROM communique électriquement avec le Z80, et pouvoir appliquer cette connaissance à votre propre système quelle que soit votre application.

1.1 - Qu'est-ce qu'une ROM ?

Dans la plupart des systèmes à microprocesseur, il y a une certaine partie de la mémoire qui peut stocker les informations et les conserver une fois l'alimentation coupée. Ce type de mémoire s'appelle *mémoire à lecture seule* ou *ROM* (*Read-Only Memory*). Les informations contenues dans une ROM peuvent être lues, mais ne peuvent pas être modifiées. Une ROM est utile dans un système parce qu'elle permet à l'Unité Centrale de traitement ou CPU (*Central Processing Unit*) d'initialiser tous les circuits périphériques en les mettant dans les états logiques voulus dès la mise sous tension.

Il y a plusieurs types de mémoires non volatiles qui peuvent être employées dans un système à microprocesseur. Ils comprennent les mémoires à lecture seule (ROM), les mémoires à lecture seule programmables (PROM), les mémoires à lecture seule programmables effaçables (EPROM) et les mémoires à lecture seule modifiables électriquement (EAROM). (Voir Figure 1.1.) Dans un système classique à Z80, on n'emploie normalement qu'un seul type de mémoire non volatile. Comme il se peut, cependant, que votre application nécessite n'importe laquelle de ces mémoires, nous allons maintenant les décrire toutes :

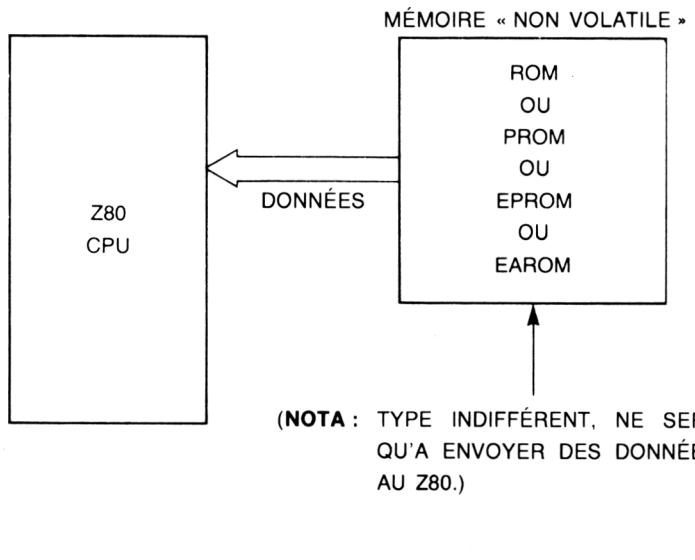


Figure 1.1 : *Schéma montrant l'application type d'une mémoire non volatile pouvant être utilisée dans un système à microprocesseur. La ROM non volatile contient habituellement des programmes d'initialisation qui sont exécutés dès la mise sous tension du système.*

ROM Avec les ROM ou *mémoires à lecture seule*, les données sont mises dans la mémoire par le fabricant. Les ROM sont employées chaque fois qu'il y a une grande quantité de données fixes. (*Note :* Mettre les données dans une ROM est une procédure très coûteuse.)

PROM Avec les PROM ou *mémoires à lecture seule programmables*, les données sont mises dans la mémoire par l'utilisateur. Des impulsions de tension élevée font, en fait, fondre des bandes de métal ou de silicium polycristallin à l'intérieur du circuit intégré, forçant des 1 et des 0 logiques dans des emplacements d'adresses spécifiques de la mémoire. Une fois introduites, les données ne peuvent pas être changées. Ce type de mémoire fonctionne habituellement à une vitesse beaucoup plus grande que les autres mémoires programmables.

EPROM Avec les EPROM ou *mémoires à lecture seule programmables effaçables*, les données sont introduites dans la mémoire par l'utilisateur au moyen de signaux à tension élevée. Ceci est semblable à la programmation d'une PROM. Les données peuvent être effacées en

projetant un rayonnement ultraviolet par une fenêtre transparente qui recouvre le circuit intégré. Après un certain temps d'exposition à ce rayonnement, toutes les données sont effacées et le boîtier peut ensuite être reprogrammé avec des données entièrement nouvelles. On utilise souvent ce type de mémoire pour des mises au point au cours desquelles le programme risque de subir de nombreuses modifications.

EAROM Avec les EAROM ou *mémoires à lecture seule modifiables électriquement*, les données sont mises dans la mémoire par l'utilisateur comme avec les EPROM. La différence majeure est que les données d'une EAROM peuvent être effacées électriquement, sans lumière ultraviolette.

Quelle que soit la mémoire non volatile utilisée dans votre système, les caractéristiques de fonctionnement sont toutes très semblables. Nous allons maintenant examiner les paramètres importants de ce groupe de mémoires. (*Note :* Dans tout le texte nous utiliserons le mot ROM pour désigner n'importe laquelle des mémoires non volatiles que nous venons de décrire.)

1.2 - Principales caractéristiques des ROM

Examinons maintenant les principales caractéristiques des ROM. Pour commencer, il est important de noter qu'une ROM ne peut être que *lue* par le CPU, d'où le nom de *mémoire à lecture seule.* En second lieu, l'information en mémoire est recherchée chaque fois qu'une adresse est appliquée aux lignes d'entrées d'adresses. Le nombre de lignes d'entrées d'adresses que possède une ROM dépend de l'organisation interne des données en mémoire.

On peut connaître la structure interne des données en mémoire en notant la spécification de la ROM. L'organisation matérielle d'une ROM peut, par exemple, être 1024×8 ou 2048×8 ou 4096×8, pour ne citer que quelques types de structures matricielles de mémoire. Le premier nombre de ces spécifications indique la quantité d'emplacements d'adresses distincts contenus dans le circuit intégré. Le second nombre indique la quantité de bits de données en parallèle pouvant être lues dans la ROM à chaque emplacement d'adresse distinct.

La remarque suivante permet de déterminer le nombre de lignes d'adresses existant dans une ROM donnée. Pour commencer, la description du boîtier, par exemple 2048×8, nous permet déjà de savoir que 2048 indique le nombre d'emplacements d'adresses distincts ; ce nombre est égal au nombre de différentes combinaisons binaires permises par les lignes d'adresses. En d'autres termes, $2^x = 2048$ donne $x = 11$ et x est égal au nombre de lignes d'adresses. Par conséquent, si la mémoire est une

4096 × 8, le nombre de lignes d'adresses est $2^x = 4096$ soit x = 12. Comme on peut le voir, cette équation est vérifiée quel que soit le nombre d'emplacements d'adresses dans la ROM.

Il est à noter que, lorsque l'on décrit une ROM, le nombre exact d'emplacements d'adresses est généralement abrégé à mille près et représenté par la simple lettre K comme Kilo. En d'autres termes, une ROM 1024 × 8 sera appelée ROM 1 K. Une ROM 4096 × 8 sera appelée ROM 4 K, etc.

Il faut enfin indiquer, en ce qui concerne le fonctionnement des ROM, que toutes les données sont lues dans la ROM en mode parallèle. Autrement dit, quand les sorties de données sont validées électriquement dans le bus de données du système, les bits de données sont pris en compte par le microprocesseur dans l'un de ses registres internes.

La Figure 1.2 donne le schéma de fonctionnement d'une ROM. Dans cette figure, on peut voir une entrée appelée *sélection de boîtier* (*chip select*)

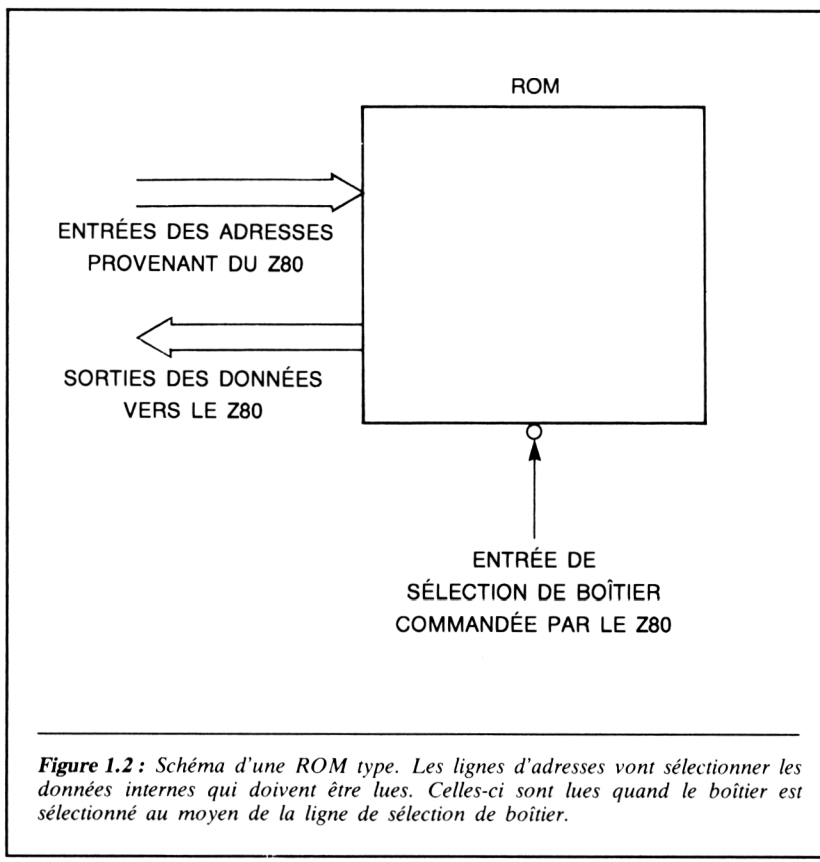


Figure 1.2 : *Schéma d'une ROM type. Les lignes d'adresses vont sélectionner les données internes qui doivent être lues. Celles-ci sont lues quand le boîtier est sélectionné au moyen de la ligne de sélection de boîtier.*

dont nous n'avons pas encore parlé. La fonction de cette ligne d'entrée est de mettre en service ou hors service les lignes de sortie de données de la ROM. Quand cette ligne de sélection de boîtier est à l'état actif, les sorties de données de la ROM sont actives et sont à l'état logique 1 ou 0, suivant la programmation de la mémoire. Quand la ligne de sélection de données n'est pas active, les sorties de données sont mises en *troisième état* (*tri-state*) ou état de haute impédance. En d'autres termes, la ligne d'entrée de sélection de boîtier autorise ou interdit électriquement les sorties de la ROM. Plus loin dans ce chapitre, nous montrerons comment utiliser cette ligne.

1.3 - Déroulement des opérations lors d'une lecture des données dans une ROM

Examinons le schéma de la Figure 1.2. Une suite d'événements électriques (énumérés ci-dessous) se déroule chaque fois que l'on doit lire des données. Gardons présent à l'esprit que cette séquence doit se dérouler quel que soit le type de CPU utilisé dans l'application concernée. Examinons cette séquence. (Nous étudierons plus tard, d'une façon plus spécifique, comment le Z80 suit ce déroulement.)

1. Une adresse est envoyée à la ROM par le CPU. Cette adresse indique l'emplacement interne de la donnée qui doit être lue. Il y a des milliers d'octets de données stockés dans la ROM. Les lignes d'adresses choisissent un seul octet de données pour le sortir.

2. Le CPU attend ensuite pendant un certain temps, appelé *temps d'accès*, d'environ 100 à 300 nanosecondes suivant le type de ROM employé, permettant ainsi à la mémoire de décoder l'adresse qui vient d'être introduite et aux données de sortie d'atteindre les lignes de sortie de données de la mémoire.

3. La sélection de boîtier est activée pour permettre aux sorties de données d'accéder au bus de données du système. A ce moment-là, les données provenant de la ROM sont présentes sur le bus de données du système et sur la broche d'entrée de données du CPU. Celui-ci prend ensuite en compte les données dans un registre interne.

4. La sélection de boîtier est désactivée pour séparer les données provenant de la ROM du bus de données du système.

Cette séquence générale se déroule chaque fois que le CPU lit des données dans la ROM. La Figure 1.3 montre un diagramme des temps pour cette séquence.

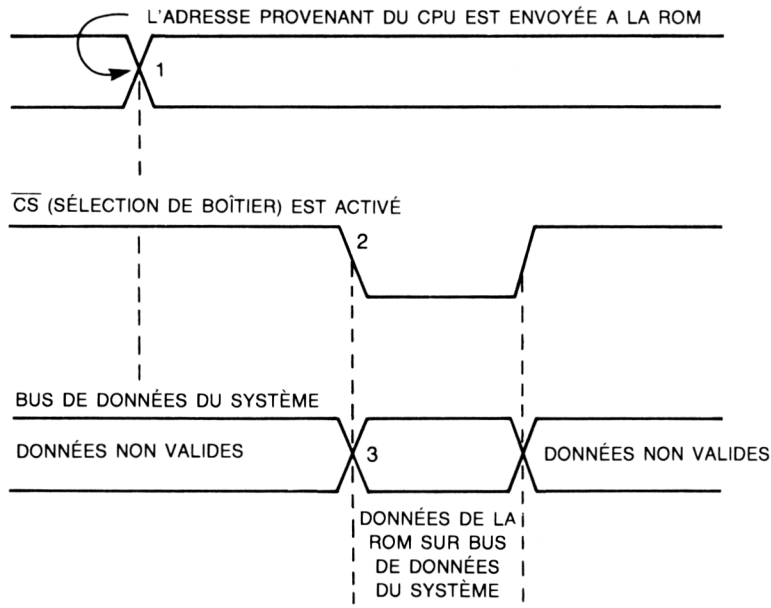


Figure 1.3 : *Diagramme général des temps pour la lecture des données dans une ROM :*

1. *L'adresse est envoyée à la mémoire par le CPU.*
2. *La sélection de boîtier est activée.*
3. *Les sorties de données de la ROM sont validées dans le bus de données.*

Les CPU sont conçus pour fonctionner dans ces limites. La plupart du temps, l'utilisateur n'a donc pas à s'occuper de cette séquence car le microprocesseur le fait à l'aide de circuits internes de temporisation. Il est néanmoins important que vous compreniez ce déroulement car il rend beaucoup plus simples l'utilisation des ROM ainsi que la compréhension du fonctionnement des circuits.

1.4 - Connexion des bus du Z80

Relions maintenant le Z80 à une ROM. La Figure 1.4 montre les liaisons physiques des lignes de données et d'adresses du Z80 à la ROM. Ces liaisons sont tout à fait directes. Remarquez que dans la Figure 1.4, la ligne de sélection de boîtier de l'EPROM 2716 n'est pas connectée. C'est parce que nous voulons analyser d'abord le *plan d'adressage*.

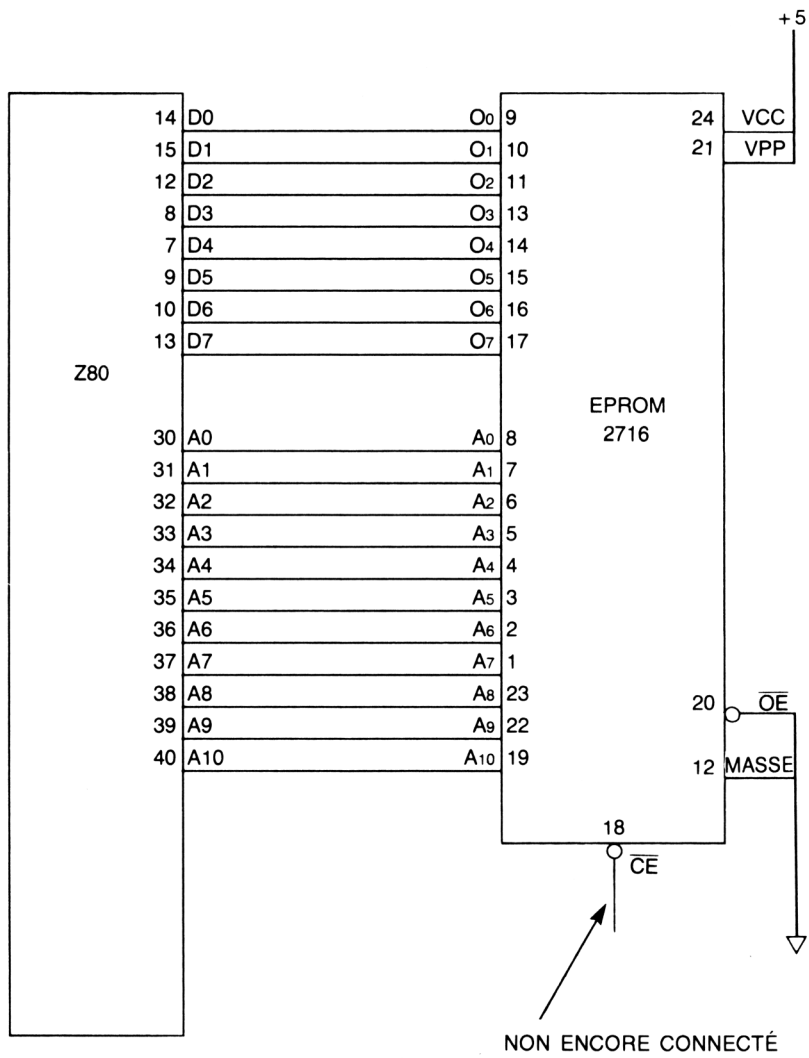


Figure 1.4 : *Plan montrant les liaisons des lignes d'adresses et de données entre un microprocesseur Z80 et une EPROM 2716 utilisant des lignes de données et d'adresses sans amplificateur de ligne.*

1.5 - Plan d'adressage

Le Z80 a 16 lignes physiques de sorties d'adresses, référencées A0 à A15. Cela signifie que le Z80 peut accéder directement à un total de 2^{16} ou 65536 emplacements de stockage séparés. Par exemple, une EPROM 2716 possède 2048 emplacements physiques d'adresses accessibles. Il faut donc une méthode pour choisir seulement 2048 emplacements parmi un total de 65536 ou 64 K possibilités. En d'autres termes, quels sont les 2048 emplacements que nous allons choisir ? Nous avons le choix entre 65536 divisé par 2048, soit 32 blocs de 2048 emplacements.

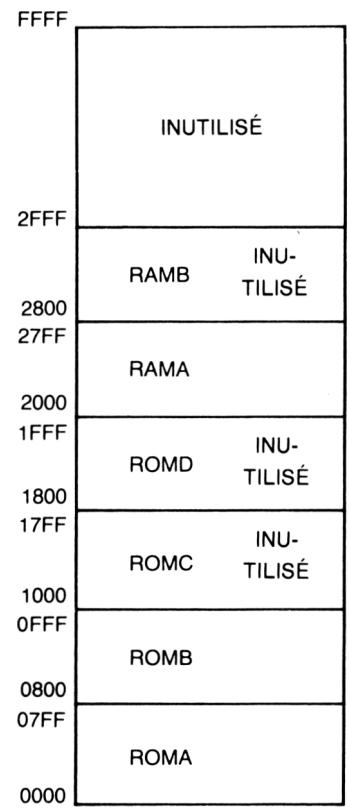


Figure 1.5 : *Plan d'adressage de mémoires pour système type à microprocesseur. Avant de concevoir un circuit, on élabore normalement un plan d'adressage des mémoires qui définit l'attribution de l'espace d'adressage.*

Avant de réaliser les circuits d'un système, le concepteur doit d'abord élaborer *un plan de mémoire*. Un plan de mémoire sert à définir les emplacements d'adresses attribués à chaque ROM, RAM, et boîtier d'entrée ou de sortie. La Figure 1.5 montre un plan d'adressage type. On peut voir dans cette figure que la totalité des 65536 emplacements d'adresses est divisée en blocs fonctionnels. Ces blocs indiquent les emplacements d'adresses réservés aux différentes ROM et RAM spécifiques.

Note : Comme le montre la Figure 1.5, l'adresse ayant la valeur la plus basse dans un système avec Z80 (0000 en hexadécimal) est habituellement réservée à la ROM. C'est parce que chaque fois qu'un Z80 est initialisé par commande extérieure ou par circuit d'initialisation à la mise sous tension, le bus d'adresses prévoit toujours que le premier code opération d'instruction se trouve à l'adresse hexadécimale 0000.

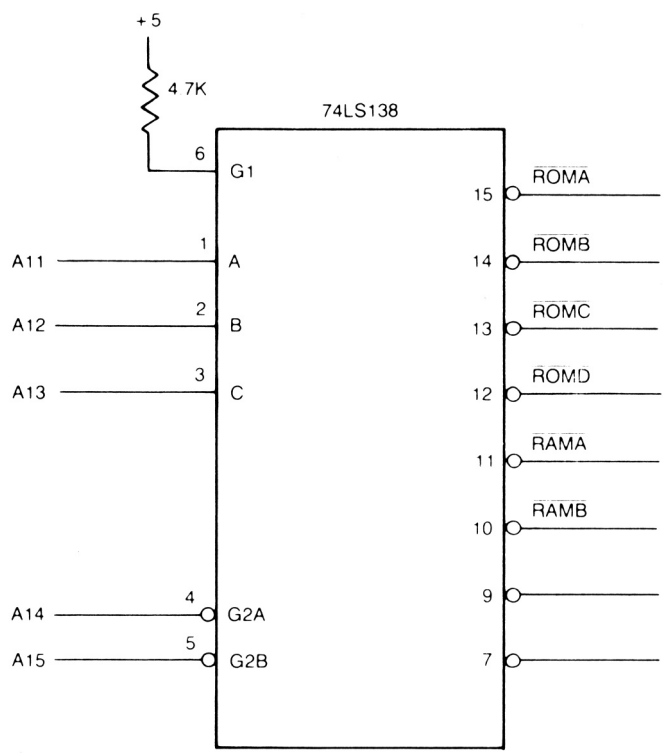


Figure 1.6 : *Circuit réalisant une partie du plan mémoire de la Figure 1.5.*

1.6 - Production des signaux de sélection de boîtier pour ROM

En se basant sur les renseignements de la Figure 1.5, on peut voir que ROMA correspond aux adresses comprises entre 0000 et 07FF incluse, envoyées par le Z80. Il faut qu'il y ait dans le système un signal électrique qui indique que l'adresse envoyée dans le bus d'adresses est comprise entre ces limites. La Figure 1.6 montre un circuit qui réalise cela.

Dans le circuit de la Figure 1.6, chaque fois que A11, A12, A13, A14 et A15 sont au 0 logique, la broche de sortie 15 du 74LS138 est au 0 logique. La broche 15 reste au 0 logique entre les limites d'adresses spécifiées. La Figure 1.7 montre comment les broches de la sortie du 74LS138 répondent aux adresses envoyées dans le bus d'adresses.

On peut voir dans la Figure 1.7 que lorsque A11 va au 1 logique, la broche 14 du 74LS138 va au 0 logique et la broche 1 va au 1 logique. Le résultat est que la broche 1 est à un 0 logique si, et seulement si, l'adresse envoyée par le Z80 est dans les limites 0000 à 07FF. Le même raisonnement

A15	A14	A13	A12	A11	A10 ---------- A0	HEX	BROCHE # = 0
0	0	0	0	0	0 ---------- 0	0000	15
0	0	0	0	0	1 ---------- 1	07FF	15
0	0	0	0	1	0 ---------- 0	0800	14
0	0	0	0	1	1 ---------- 1	0FFF	14
0	0	0	1	0	0 ---------- 0	1000	13
0	0	0	1	0	1 ---------- 1	17FF	13
0	0	0	1	1	0 ---------- 0	1800	12
0	0	0	1	1	1 ---------- 1	1FFF	12
0	0	1	0	0	0 ---------- 0	2000	11
0	0	1	0	0	1 ---------- 1	27FF	11
0	0	1	0	1	0 ---------- 0	2800	10
0	0	1	0	1	1 ---------- 1	2FFF	10

Figure 1.7 : *Plan d'adressage des mémoires pour la Figure 1.6.*

peut être appliqué aux autres broches de sortie indiquées dans le tableau de la Figure 1.7.

Il est important d'avoir présent à l'esprit que le circuit montré Figure 1.6 n'est qu'une manière parmi beaucoup d'autres de choisir un bloc d'adresses donné dans l'espace d'adressage existant.

1.7 - Production des signaux de lecture de mémoire

Nous sommes maintenant prêts à connecter au Z80 la broche d'entrée de sélection de boîtier. Nous devons cependant nous renseigner d'abord sur le signal de commande synchronisé issu du Z80 qui déclenche et arrête le transfert de données. Ce signal s'appelle *lecture de mémoire* (*MEMory Read*, ou signal $\overline{\text{MEMR}}$). Le signal $\overline{\text{MEMR}}$ et le signal de sélection de mémoire fonctionnent ensemble pour autoriser les sorties de données de la ROM dans le bus de données du système à l'instant approprié. La Figure 1.8 montre une manière permettant au Z80 de produire les signaux $\overline{\text{MEMR}}$.

Dans la Figure 1.8, la ligne $\overline{\text{MREQ}}$ est mise au 0 logique chaque fois que le Z80 est en train de communiquer avec la mémoire. Les instructions du logiciel dictent cette action. $\overline{\text{RD}}$ est un signal de commande temporisé issu du Z80. Il devient un 0 logique chaque fois que le Z80 est préparé

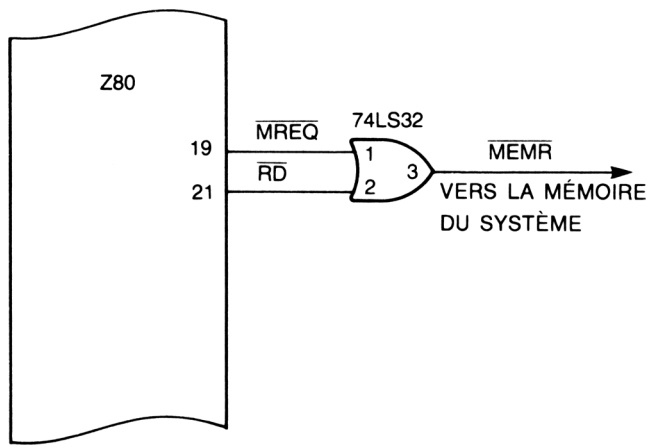


Figure 1.8 : *Schéma montrant comment le signal de commande de lecture de la mémoire du système est produit en effectuant un OU logique de signaux de commande provenant du microprocesseur Z80.*

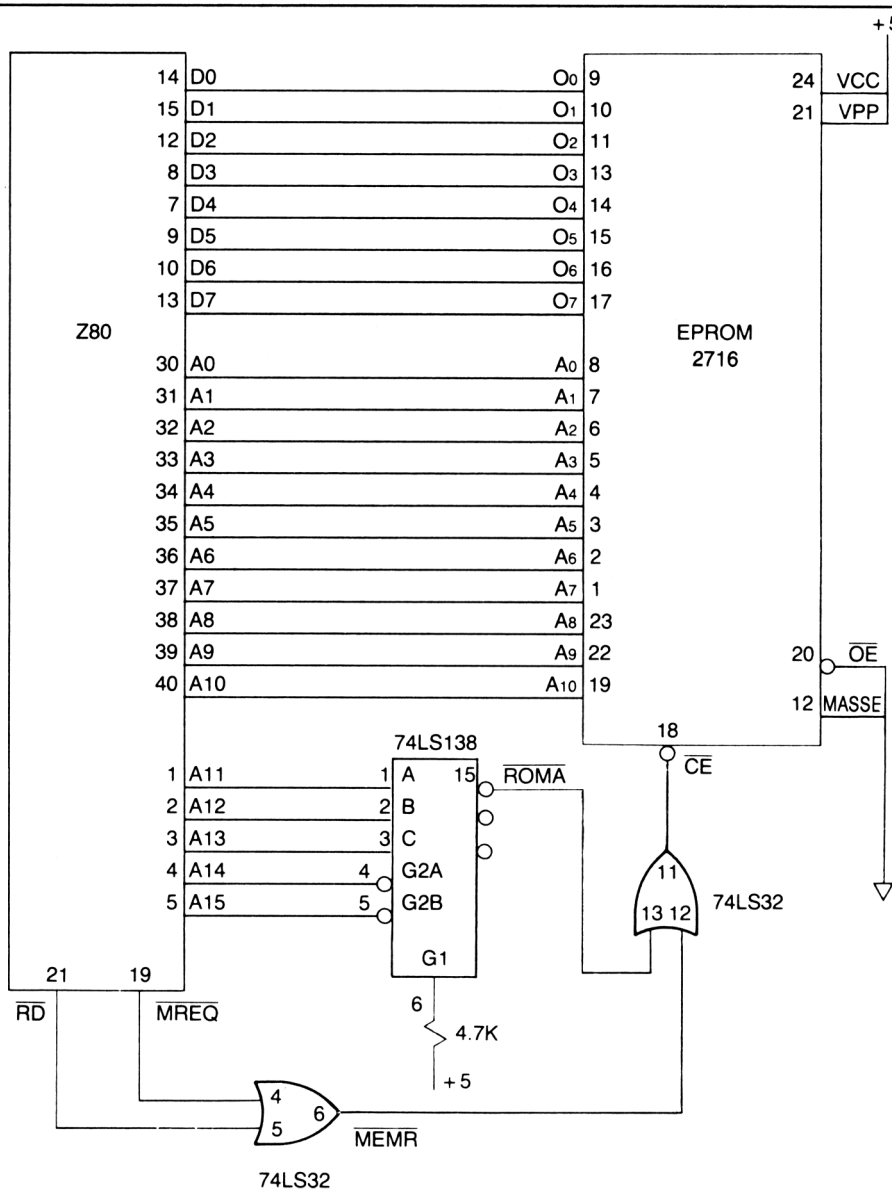


Figure 1.9 : *Schéma complet montrant les connexions entre EPROM 2716 et microprocesseur Z80. Ce schéma comporte les éléments logiques employés pour activer la mémoire conformément au plan d'adresses.*

électriquement à recevoir des données de la mémoire ou d'un dispositif d'entrée-sortie.

1.8 - Connexion des lignes de sélection de boîtier (*chip select*)

La Figure 1.9 montre le schéma complet de la liaison d'un Z80 à une ROM. Quand les signaux de sélection de mémoire (*memory select*) et de lecture de mémoire (*memory read*) sont tous les deux au 0 logique, l'entrée de sélection de boîtier de l'EPROM 2716 est active à l'état logique 0. La Figure 1.10 montre le diagramme des temps pendant une opération de lecture de ROM, et les importantes relations existant entre les signaux.

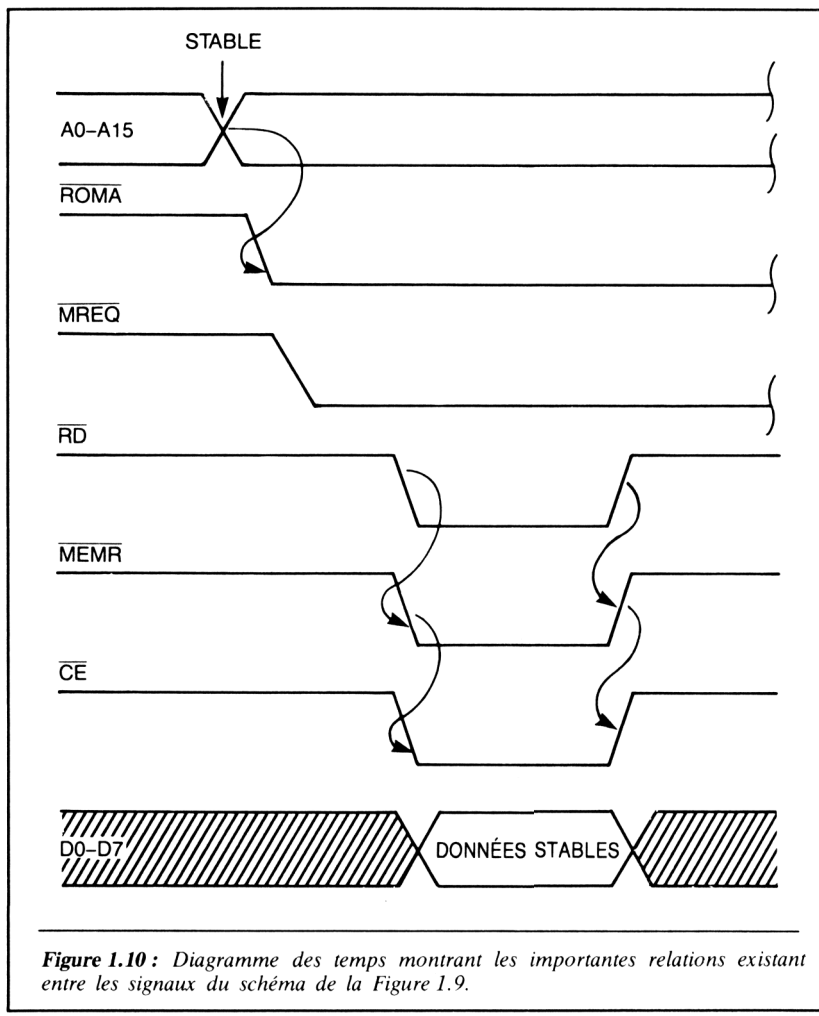


Figure 1.10 : *Diagramme des temps montrant les importantes relations existant entre les signaux du schéma de la Figure 1.9.*

1.9 - Variante

La Figure 1.11 montre une autre technique que l'on peut employer pour valider les données de sortie de la ROM dans le bus de données du système au moment approprié. Cette technique utilise le décodage interne effectué à l'intérieur du boîtier 2716. Dans ce cas, les broches 20 $\overline{OE}$ et 18 $\overline{CE}$ doivent être au 0 logique avant que les sorties de données ne soient activées dans le bus de données du système. Rappelons qu'avec la technique précédente, la broche d'entrée 20 $\overline{OE}$ était reliée à la masse constituant l'état actif pour cette entrée.

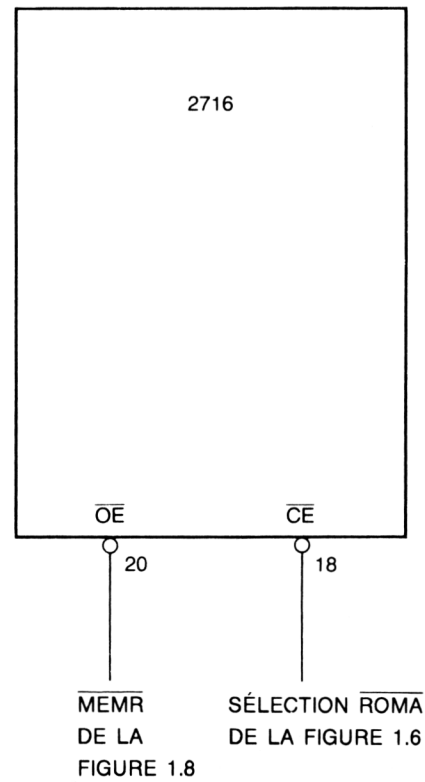


Figure 1.11 : *Schéma montrant comment le boîtier 2716 peut être activé en utilisant la broche de validation 20 ou la broche de sélection de boîtier 18.*

1.10 - Addition de plusieurs ROM

La Figure 1.12 montre qu'il est possible d'ajouter des ROM au système en employant les techniques de validation de la Figure 1.11. Cette technique permet d'ajouter davantage de boîtiers de ROM sans avoir à ajouter de dispositifs logiques extérieurs. (Rappelons que la technique de décodage exposée dans la Figure 1.10 nécessite des opérateurs logiques supplémentaires quand on ajoute d'autres ROM au système.)

Rappelons-nous aussi qu'un bloc d'adresses spécifique est réservé à chaque ROM. Cependant le Z80 ne fait pas de distinction quand il exécute le logiciel. Pour lui, tout l'espace d'adressage des ROM est disponible. En d'autres termes, une instruction à trois octets peut comporter un octet situé dans le dernier emplacement de ROMA et les deux autres octets dans les premiers emplacements de ROMB.

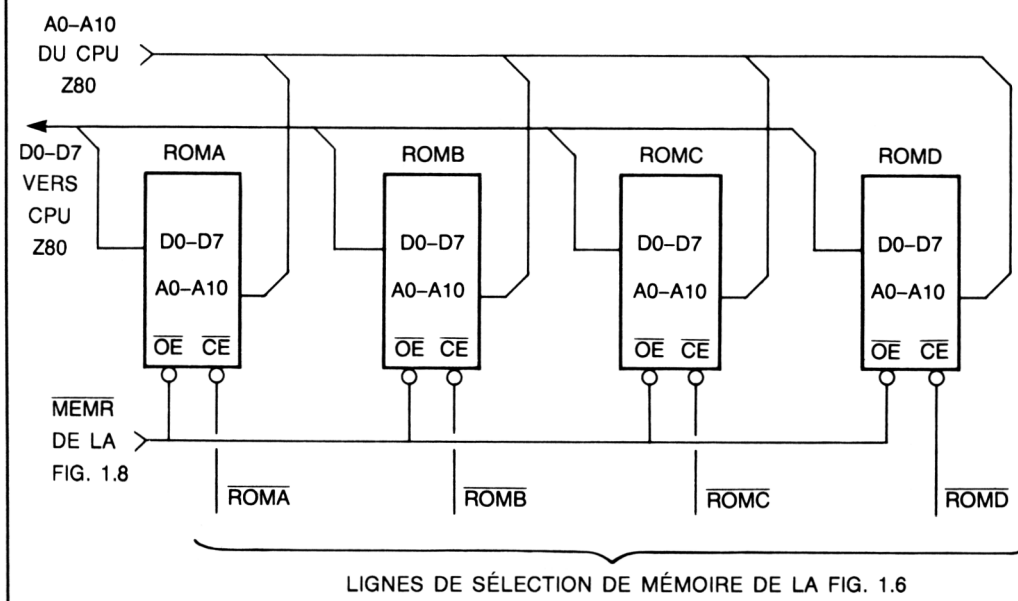


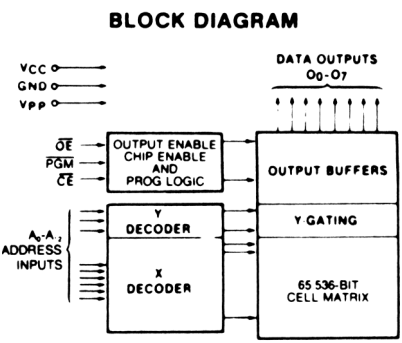
Figure 1.12 : *Schéma montrant comment on peut ajouter plusieurs boîtiers de mémoire au circuit sans éléments logiques supplémentaires. La ligne $\overline{OE}$ du 2716 intervient dans cette technique de décodage de mémoire.*

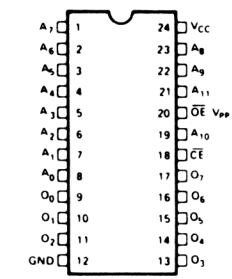
1.11 - Etablissement d'un plan d'adressage pour ROM plus importantes

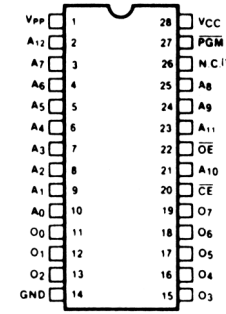
Il y a deux ROM plus importantes qui sont très employées pour réaliser des systèmes : la 2732 et la 2764. Ces mémoires sont organisées en 4096 × 8 et 8192 × 8, respectivement. La Figure 1.13 montre leurs brochages.

Un point intéressant en ce qui concerne les brochages est que ces deux boîtiers sont interchangeables dans le même support, bien que leurs nombres de broches soient différents. On obtient cela en utilisant des supports à 28 contacts et en connectant Vcc aux contacts 28 et 26. (Voir Figure 1.14.)

Ces boîtiers sont conçus pour permettre de commencer avec une 2732, puis de passer ultérieurement à la 2764. Ceci pour doubler l'espace ROM







[1] For total compatibility and upgradability from the 2732A and ROMs provide a trace to pin 26.

MODE SELECTION

MODE \ PINS	CE (20)	OE (22)	PGM (27)	V_{PP} (1)	V_{CC} (28)	Outputs (11-13, 15-19)
Read	V_{IL}	V_{IL}	V_{IH}	V_{CC}	V_{CC}	D_{OUT}
Standby	V_{IH}	x	x	V_{CC}	V_{CC}	High Z
Program	V_{IL}	x	V_{IL}	V_{PP}	V_{CC}	D_{IN}
Program Verify	V_{IL}	V_{IL}	V_{IH}	V_{PP}	V_{CC}	D_{OUT}
Program Inhibit	V_{IH}	x	x	V_{PP}	V_{CC}	High Z

x can be either V_{IL} or V_{IH}.

PIN NAMES

A_0-A_{12}	ADDRESSES
CE	CHIP ENABLE
OE	OUTPUT ENABLE
O_0-O_7	OUTPUTS
PGM	PROGRAM
N.C	NO CONNECT

*HMOS is a patented process of Intel Corporation.

Figure 1.13 : *Brochage et schéma des EPROM 2732 et 2764 très utilisées dans les applications à microprocesseur.*

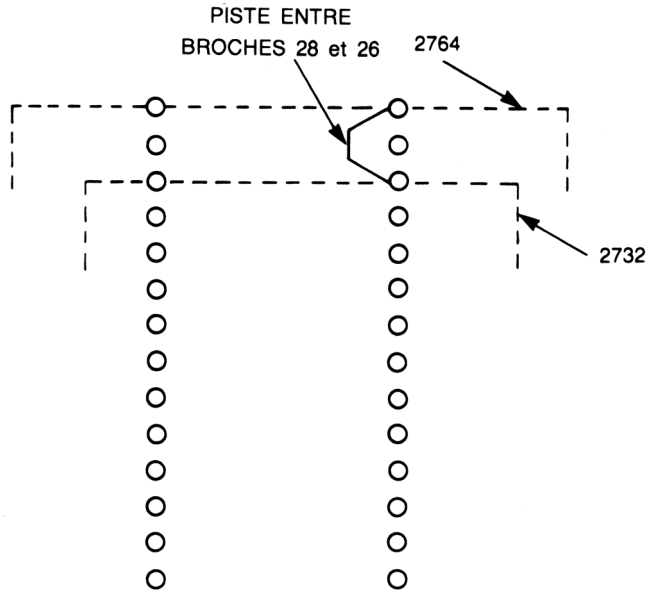


Figure 1.14 : *Schéma montrant comment un seul support de circuit intégré peut servir soit au boîtier 2732, soit au boîtier 2764. Dans ce cas, on a utilisé un support à 28 contacts.*

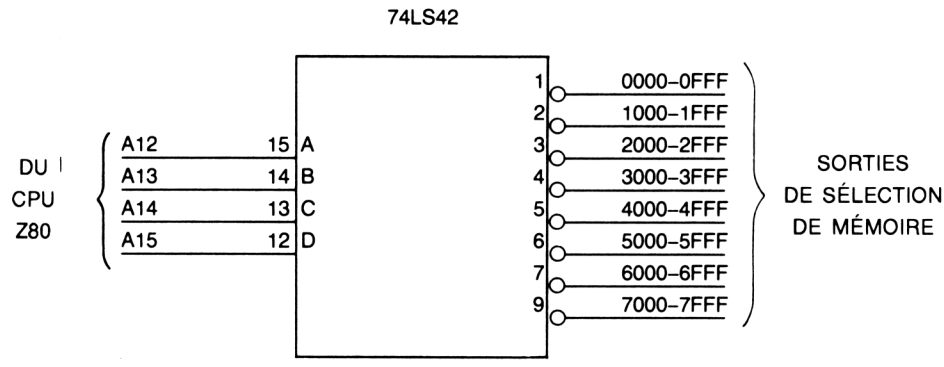


Figure 1.15 : *Circuit montrant la logique de décodage de sélection de mémoire consistant en un décodeur BCD/décimal 74LS42.*

existant sans avoir à agrandir physiquement les circuits. Si on choisit des mémoires plus importantes, le plan d'adressage passera de celui montré en Figure 1.5 à celui montré en Figure 1.15. Notez que dans la Figure 1.15 on utilise seulement un boîtier pour l'espace d'adressage allant de 0000 à 0FFF. (Rappelons-nous que la Figure 1.5 en exige deux.) Les circuits utilisés pour obtenir les lignes de sélection de mémoire sont similaires à ceux utilisés précédemment pour les EPROM 2716.

1.12 - Amplificateurs de bus d'adresses

Le système de mémoire que nous avons présenté est un système de base. Il est néanmoins très employé dans l'industrie. Notons qu'il comporte un point critique : la charge des lignes de bus d'adresses du Z80.

Si nous examinons une feuille de spécifications de Z80, comme celle de la Figure 1.16, les valeurs pour I_{OL} (courant de sortie à l'état logique 0) et I_{OH} (courant de sortie à l'état logique 1) sont pour une ligne de sortie d'adresse 1,8 mA et $-250\ \mu A$, respectivement. (Le signe moins indique que le courant est fourni par le Z80, et par conséquent sort du CPU.) Cette spécification nous dit qu'une ligne d'adresse ne peut supporter qu'une seule charge TTL (*Transistor Transistor Logic*). Une charge TTL est égale à 1,6 mA à l'état logique 0 et 40 micro-ampères à l'état logique 1. (Voir Figure 1.16.)

D.C. CHARACTERISTICS
T_A = 0°C to 70°C, V_{CC} = 5V ± 5% unless otherwise specified

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNIT	TEST CONDITION
V_{ILC}	Clock Input Low Voltage	−0.3		0.8	V	
V_{IHC}	Clock Input High Voltage	Vcc−.6		Vcc+.3	V	
V_{IL}	Input Low Voltage	−0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = −250 μA
I_{CC}	Power Supply Current			150*	mA	
I_{LI}	Input Leakage Current			±10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT} = 2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float			−10	μA	V_{OUT} = 0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \leq V_{IN} \leq V_{CC}$

Figure 1.16 : *Feuille de caractéristiques partielle d'un microprocesseur Z80 donnant les courants d'entrée et de sortie pour les lignes de sortie du CPU.*

Dans les systèmes de mémoire présentés jusqu'à présent, le seul système TTL employé a été le 74LS138. LS indique *Low power Schottky* et est équivalent à environ 1/3 d'une charge TTL standard. Cette charge n'était présente que sur les lignes d'adresses de poids supérieur, A11 à A15. Toutes les autres sorties d'adresses du Z80 n'avaient pour charge que les entrées d'adresses de la mémoire. Ces dernières ont pour charge type 10 microampères aux états logiques 1 et 0. Cependant, les lignes d'adresses du système peuvent être reliées à d'autres éléments en plus de la mémoire. Elles peuvent être connectées à des dispositifs d'entrée-sortie, des câbles ou même, par des connecteurs de cartes, à d'autres cartes à circuits imprimés.

Dans ces conditions et pour toutes les applications où la charge de sortie des lignes d'adresses du Z80 excède la valeur spécifiée, des amplificateurs d'adresses seront nécessaires. Les amplificateurs d'adresses augmentent les possibilités en courant du bus d'adresses du Z80. On doit utiliser des

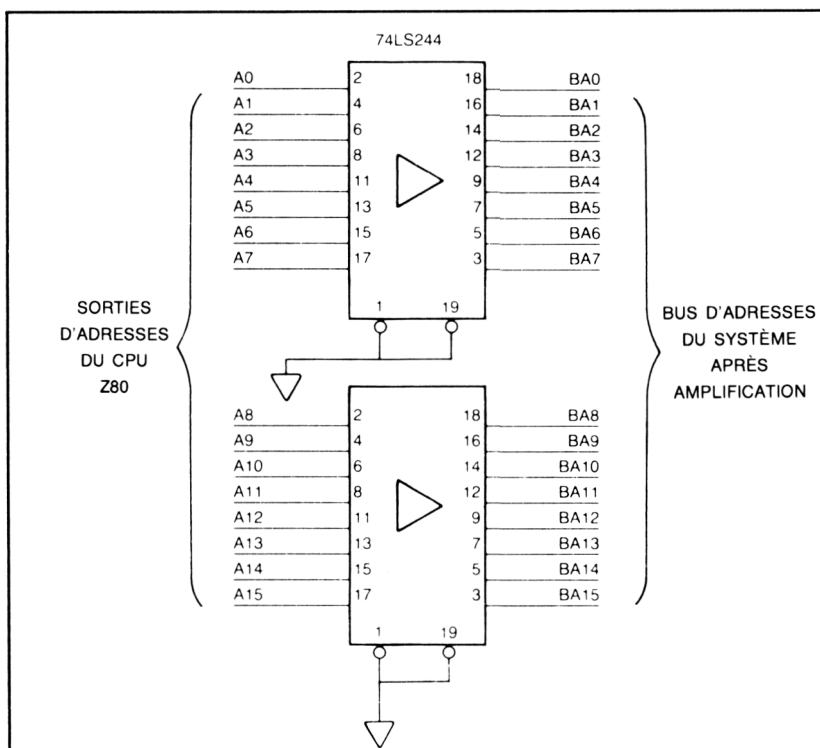


Figure 1.17 : *Schéma montrant une technique permettant d'ajouter des amplificateurs d'adresses à un système à Z80.*

amplificateurs d'adresses chaque fois que les lignes d'adresses alimentent un câble même si la charge en courant continu n'excède pas la spécification. Ceci provient de la charge capacitive due au câble. La Figure 1.17 montre un moyen pour ajouter des amplificateurs à un système à Z80. Notez cependant, que les systèmes à Z80 ne requièrent pas tous des amplificateurs d'adresses. L'utilisation d'amplificateurs dépend des applications.

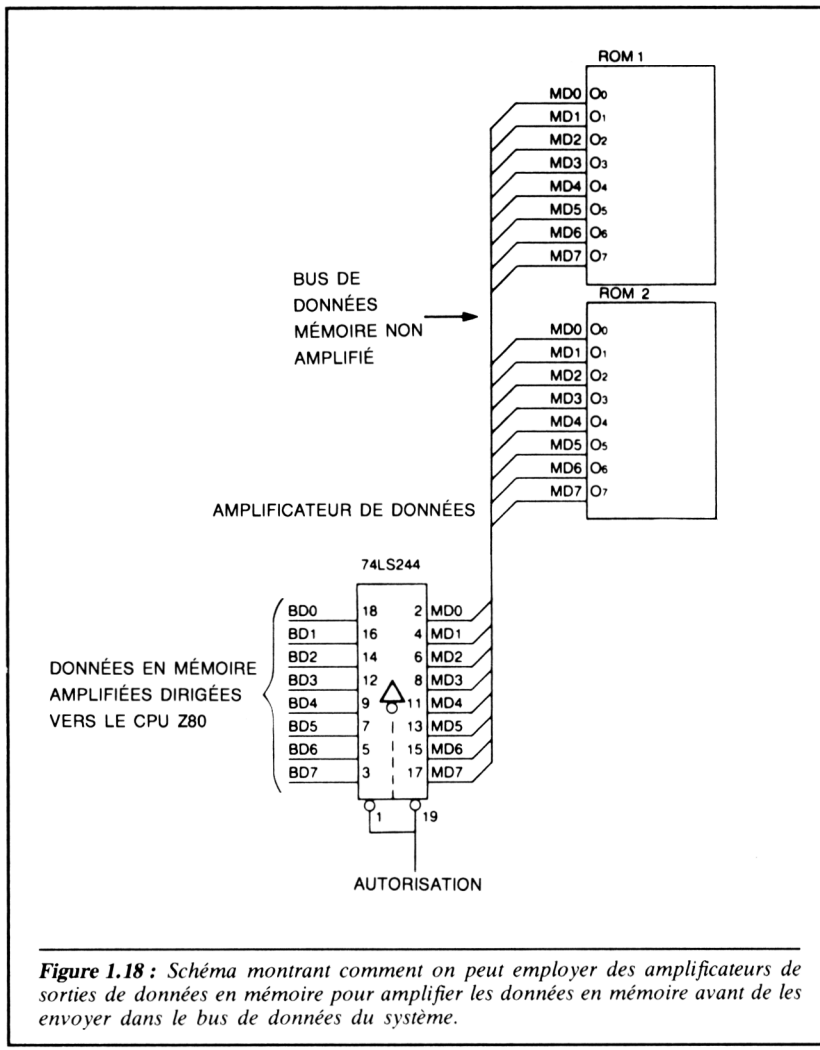


Figure 1.18 : *Schéma montrant comment on peut employer des amplificateurs de sorties de données en mémoire pour amplifier les données en mémoire avant de les envoyer dans le bus de données du système.*

1.13 - Amplification des données en mémoire

Dans certaines applications, les sorties de données des ROM n'ont pas suffisamment de possibilité de courant pour commander de façon appropriée les lignes du bus de données. Dans ce cas, il faut utiliser des

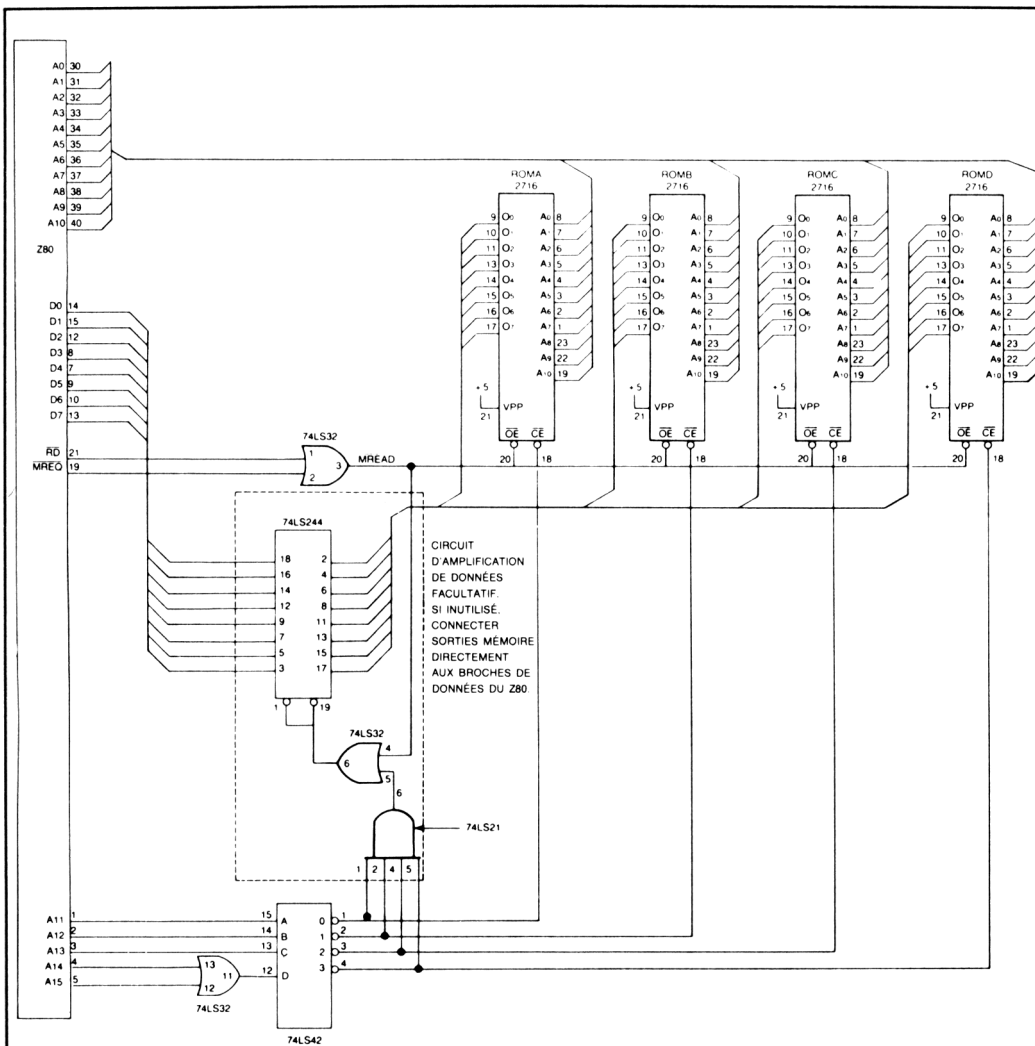


Figure 1.19 : *Système complet de mémoires utilisant le plan d'adressage de la Figure 1.22. Ce système emploie des boîtiers 2716.*

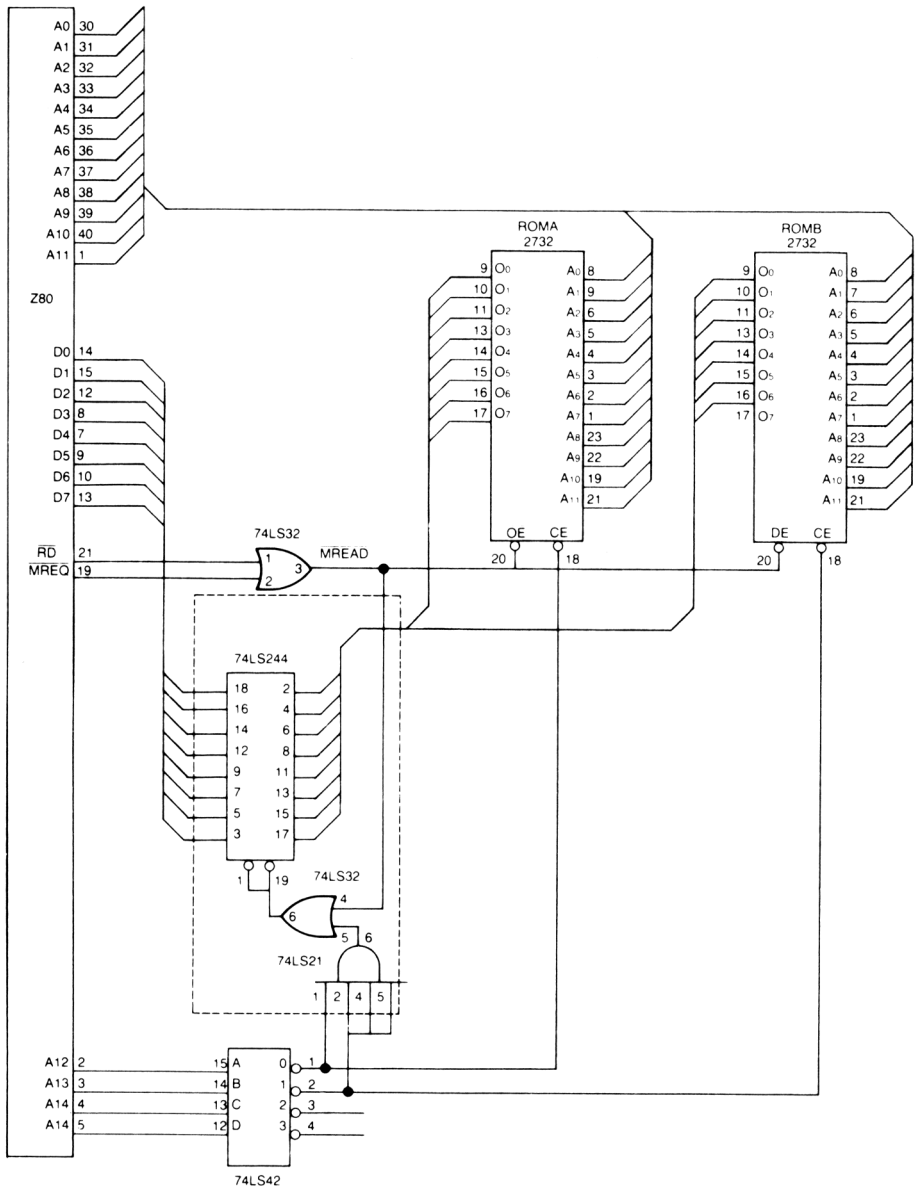


Figure 1.20 : *Système complet de mémoires utilisant le plan d'adressage de la Figure 1.22. Ce système emploie des boîtiers 2732.*

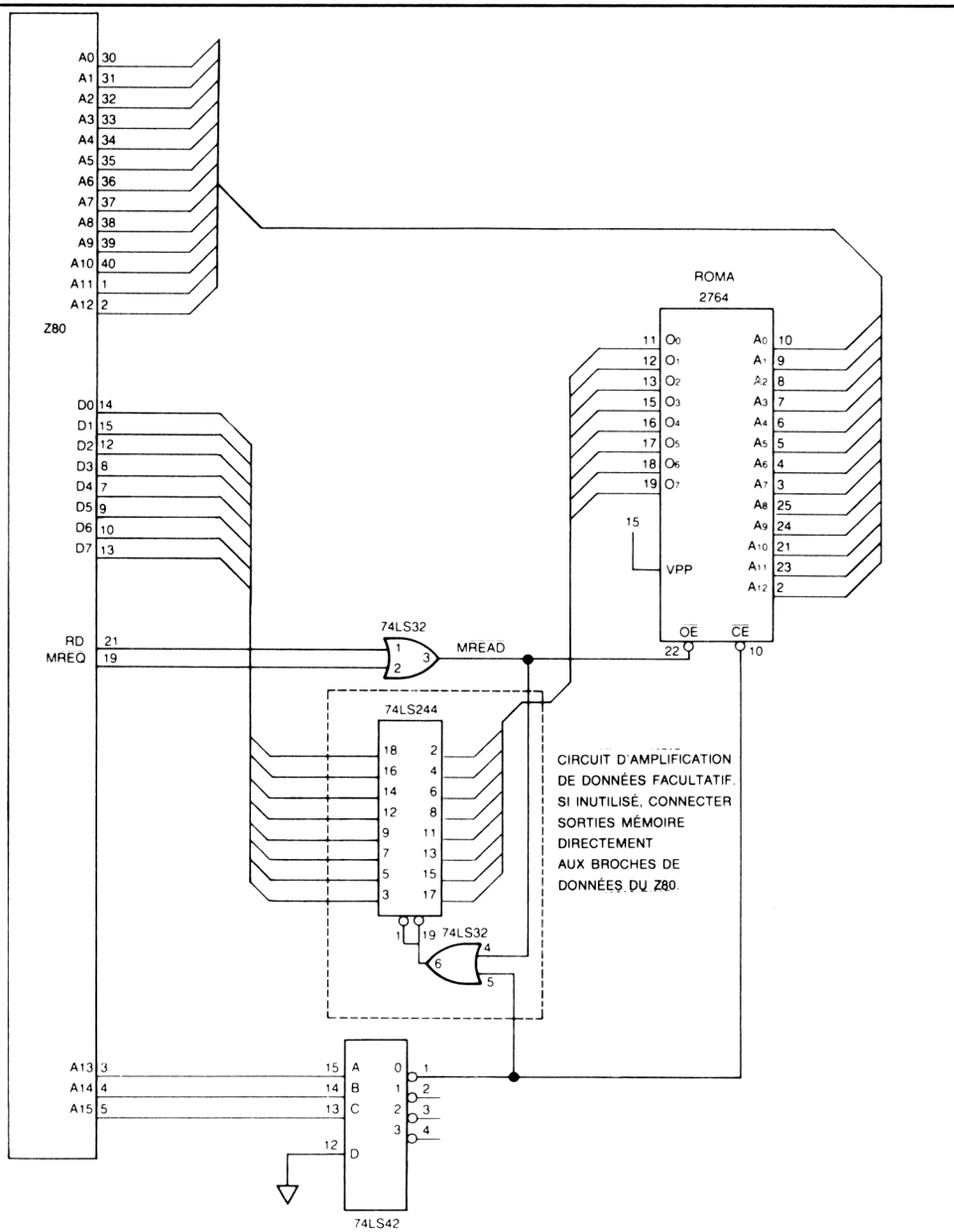


Figure 1.21 : *Système complet de mémoires utilisant le plan d'adressage de la Figure 1.22. Ce système emploie un boîtier 2764.*

amplificateurs de données. Les amplificateurs de données des mémoires effectuent une fonction similaire à celle des amplificateurs d'adresses des mémoires analysées précédemment.

Les amplificateurs de données des mémoires doivent pouvoir fonctionner en *trois états.* Un fonctionnement à trois états est nécessaire car les données en mémoire doivent être électriquement séparées du bus de données du système quand le Z80 n'en a pas électriquement besoin. La Figure 1.18 donne un schéma montrant comment les amplificateurs des données en mémoire peuvent être disposés dans un système à ROM classique. Dans cette figure, les sorties de données des ROM ont seulement à attaquer les entrées des amplificateurs de données. Les sorties des amplificateurs de données en mémoire ont pour attribution la charge complète du bus de données du système.

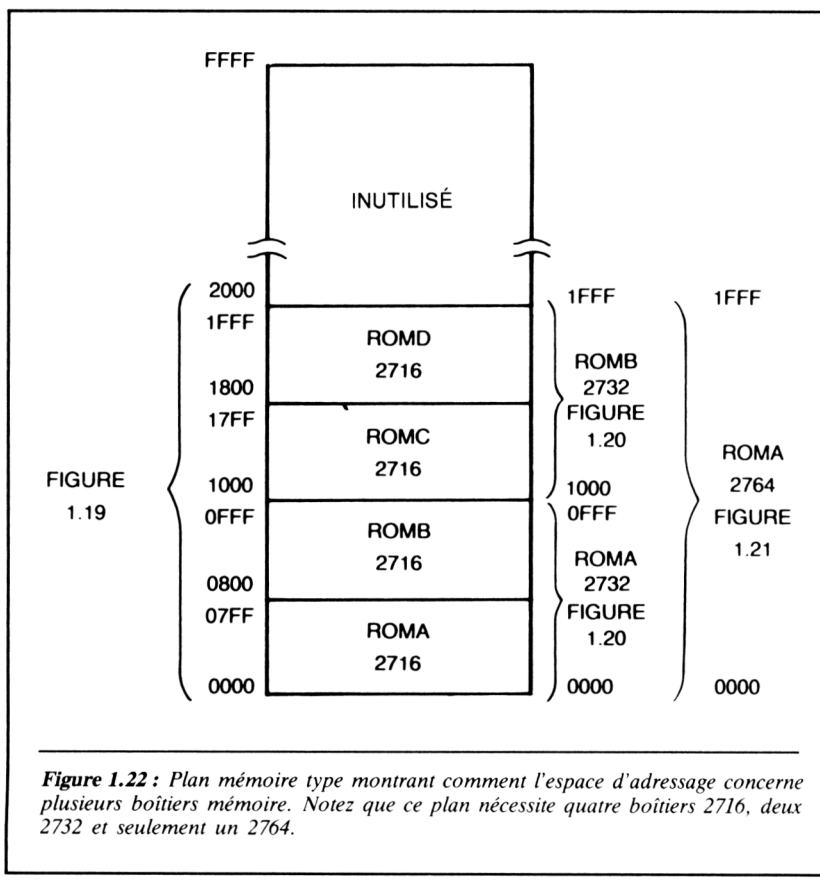


Figure 1.22 : *Plan mémoire type montrant comment l'espace d'adressage concerne plusieurs boîtiers mémoire. Notez que ce plan nécessite quatre boîtiers 2716, deux 2732 et seulement un 2764.*

1.14 - Trois exemples de systèmes à ROM complets

Les schémas des Figures 1.19, 1.20 et 1.21 sont tous des systèmes à ROM complets basés sur les notions présentées dans ce chapitre. Le plan de ces systèmes apparaît en Figure 1.22. Les systèmes de mémoires utilisant ces boîtiers sont aujourd'hui d'un usage courant. Les EPROM sont disponibles auprès de nombreux fournisseurs et sont faciles à programmer et à utiliser.

RESUME

Dans ce chapitre, nous avons donné des renseignements utiles concernant l'utilisation de ROM avec le Z80. Nous avons fourni des informations de base en ce qui concerne le fonctionnement des ROM et avons décrit les éléments fondamentaux qui permettent des liaisons valables entre Z80 et ROM. Les exemples de ce chapitre ont montré que l'emploi de ROM avec le Z80 peut être une tâche relativement simple. Il y a cependant des lignes de conduite qui doivent être suivies.

UTILISATION DE RAM STATIQUES AVEC LE Z80

CHAPITRE 2

INTRODUCTION

Nous allons maintenant poursuivre notre étude du microprocesseur Z80 en apprenant comment l'interfacer avec des *mémoires à accès aléatoire* appelées également RAM (*Random Access Memory*). On utilise une RAM dans un système pour effectuer des stockages temporaires de programmes, de données ou de variables. Contrairement à la ROM (mémoire à lecture seule, étudiée au Chapitre 1), la RAM est volatile et les informations sont perdues quand l'alimentation est coupée.

L'interface électrique entre Z80 et RAM statique n'est pas difficile à réaliser. Il faut néanmoins suivre certaines lignes de conduite. Nous allons les examiner dans ce chapitre et passer en revue tout ce qui concerne l'interfaçage des RAM statiques. Nous montrerons et analyserons aussi deux systèmes à RAM statiques bien connus et souvent utilisés dans l'industrie. L'un d'eux comporte des mémoires 2114 de 1 K × 4 bits et l'autre des mémoires 6116 de 1 K × 8 bits.

2.1 - Aperçu sur la manière de communiquer avec des RAM statiques

Nous allons commencer notre analyse en expliquant comment on communique, en général, avec une mémoire à semi-conducteur à lecture et écriture. Une fois que vous aurez compris ce processus, vous saisirez très facilement comment le Z80 communique avec une RAM. Si vous le savez déjà, vous pouvez, si vous le désirez, sauter la première partie de ce chapitre.

Pour commencer, la mémoire à accès aléatoire d'un système à microprocesseur permet deux choses : on peut, au moyen du CPU, y écrire des données, et on peut lire les données qu'elle contient. Des signaux électriques spécifiques sont nécessaires pour effectuer ces deux types de communications. La Figure 2.1 montre le schéma d'une RAM type et les principaux signaux électriques.

Examinons d'abord les liaisons d'alimentation. Une RAM doit être alimentée, et les tensions les plus répandues sont + 5 V et 0 V (masse). (Vous pouvez vous référer aux feuilles de caractéristiques du fabricant pour connaître les tensions exactes.) En ce qui nous concerne, nous supposerons que les RAM que nous utilisons nécessitent + 5 V et 0 V.

Dans la Figure 2.1, les lignes référencées Entrée des données sont les fils qui permettent aux informations d'être écrites dans la RAM. Les lignes référencées Sortie des données sont les fils qui permettent aux informations stockées dans la RAM d'être lues. Une RAM doit avoir au moins une ligne d'entrée de données et une ligne de sortie de données. Le nombre exact dépend de l'organisation interne du boîtier.

Nous avons indiqué dans le Chapitre 1 l'organisation interne d'une ROM. Une RAM peut être décrite d'une façon très similaire. Comme pour les ROM, on peut tirer beaucoup de renseignements de la description d'une RAM. On peut, par exemple, dire d'une RAM qu'elle est une 256 × 1 ou 256 × 4 ou 1024 × 8, etc. Le second nombre indique le nombre de lignes

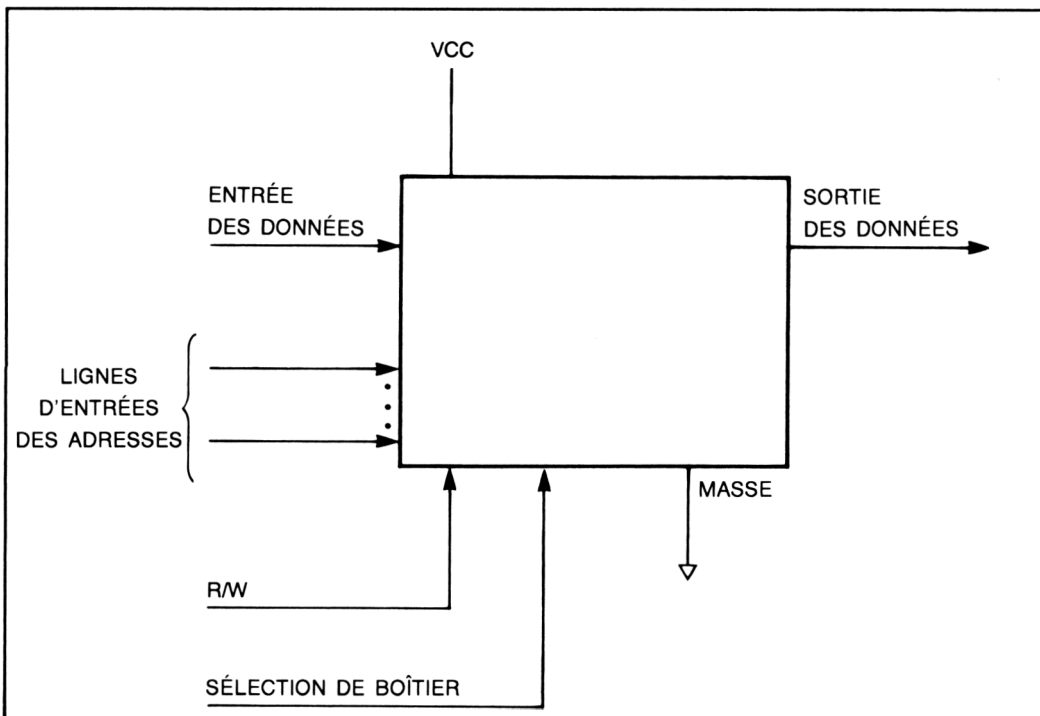


Figure 2.1 : *Schéma montrant les principaux signaux électriques d'une RAM type.*

d'entrée de données et de sortie que possède la mémoire. En d'autres termes, chaque fois qu'on écrit ou qu'on lit dans la mémoire, le second nombre détermine le nombre de bits de données parallèles qui sont écrites ou lues.

Examinons maintenant les lignes d'entrées d'adresses de la Figure 2.1. (Nous les avons déjà vues dans le Chapitre 1 au cours de notre analyse des ROM.) Les lignes d'adresses de la RAM statique ont pour fonction de choisir les emplacements internes de la mémoire où l'on doit lire ou écrire les données. On peut déterminer le nombre d'emplacements individuels de stockage d'une RAM simplement en comptant le nombre de lignes d'adresses. L'inverse est également vrai. Cela signifie que l'on peut déterminer le nombre de lignes d'adresses d'une mémoire si l'on connaît le nombre d'emplacements distincts de stockage. Par exemple, une RAM statique 1024 × 4, aura 1024, soit 1 K, emplacements mémoire distincts, et à chaque emplacement mémoire, on écrira ou lira 4 bits en même temps.

Le dernier signal référencé sur la figure (R/W) détermine si la RAM fait l'objet d'une écriture ou d'une lecture.

Examinons maintenant ce qui se passe, en général, à chaque écriture ou lecture dans une mémoire à semi-conducteur, c'est-à-dire chaque fois que le microprocesseur communique avec la RAM. Voyons d'abord ce qui se produit durant une opération de lecture.

2.2 - Déroulement des opérations pendant une lecture de RAM

1. D'abord, les adresses sont introduites dans la mémoire. A ce moment-là, l'emplacement interne où les données doivent être lues fait l'objet d'un décodage logique par la RAM.

2. La ligne de commande R/W est mise à l'état logique convenant à une lecture en mémoire. Pour certaines mémoires, c'est un 1 ; pour d'autres, c'est un 0. (Les feuilles de spécifications des fabricants vous aideront à déterminer l'état logique exact pour une lecture en mémoire dans votre système.)

3. Le système doit attendre pendant un certain laps de temps, appelé *temps d'accès lecture*, pour permettre aux circuits internes de la mémoire de décoder l'adresse et de prendre en compte les données adressées.

4. Après ce temps d'attente, les données sont disponibles sur les lignes de sortie de la mémoire et peuvent alors être lues par le microprocesseur du système. Si le microprocesseur lit les données

trop tôt, c'est-à-dire s'il ne respecte pas le temps d'accès lecture, des données non valables risquent d'être lues.

Un diagramme des temps pour cette séquence est représenté en Figure 2.2. Nous y reviendrons lorsque nous examinerons les communications entre Z80 et RAM à semi-conducteur. La séquence de lecture de données en mémoire est exactement la même pour la RAM que pour la ROM. En fait, le système à microprocesseur ne fait pas de distinction, au point de vue électrique, entre les lectures de données en ROM et les lectures en RAM.

Etudions maintenant ce qui se passe au cours de l'écriture de données en RAM.

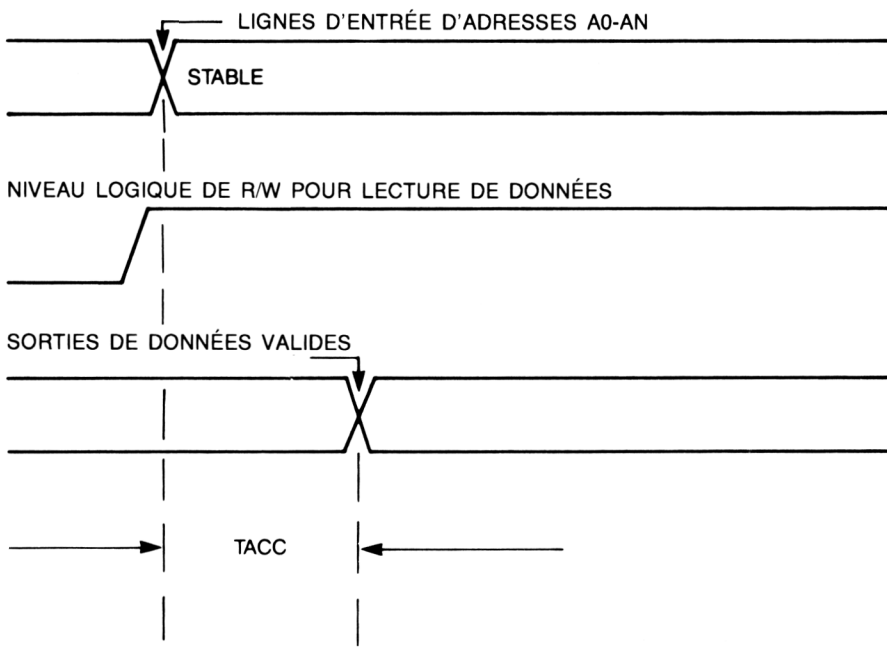


Figure 2.2 : *Diagramme des temps montrant le déroulement des opérations pour une lecture de données en mémoire.*

2.3 - Déroulement des opérations pour une écriture en RAM

1. D'abord, les lignes d'adresses de la mémoire sont mises dans les états logiques correspondant à l'emplacement interne de la mémoire dans lequel les données doivent être écrites.

2. Ensuite, les données qui doivent être écrites dans la mémoire sont mises sur les lignes de données.

3. Le système doit attendre pendant un temps déterminé, appelé *temps d'accès écriture* (habituellement inférieur à quelques centaines de nanosecondes), pour permettre aux circuits internes de décodage de se stabiliser.

4. Après ce temps d'attente, la ligne de commande R/W de la mémoire est mise au niveau logique approprié, ou reçoit une impulsion, pour permettre aux données des lignes d'entrée de données d'être écrites dans la RAM.

La Figure 2.3 montre un diagramme des temps représentant cette séquence. Les circuits nécessaires varient avec chaque microprocesseur.

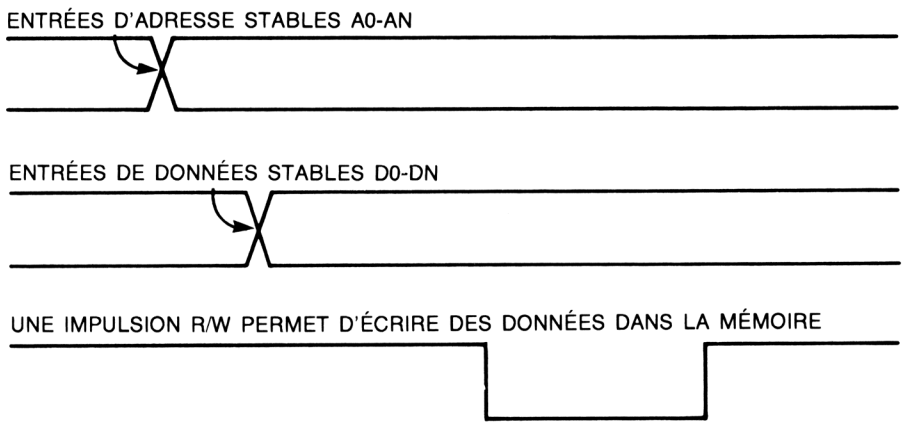


Figure 2.3 : *Diagramme des temps montrant le déroulement des opérations lors d'une écriture de données dans une RAM type.*

2.4 - Mémoire réelle

Examinons maintenant une mémoire réelle type très utilisée dans l'industrie. Nous commencerons par regarder les principales caractéristiques la concernant. Nous montrerons ensuite comment les données sont lues et écrites. Lorsque l'on utilise des RAM, il est essentiel d'en comprendre les spécifications au point de vue électrique. Une fois que vous aurez compris le fonctionnement détaillé d'une RAM, il sera facile de saisir celui d'autres RAM utilisées dans des applications différentes.

La RAM que nous avons choisie pour cette étude est la RAM statique 2114 à entrées-sorties communes et à organisation matricielle 1024 × 4. Le terme *entrées-sorties communes* s'applique à la configuration électrique des

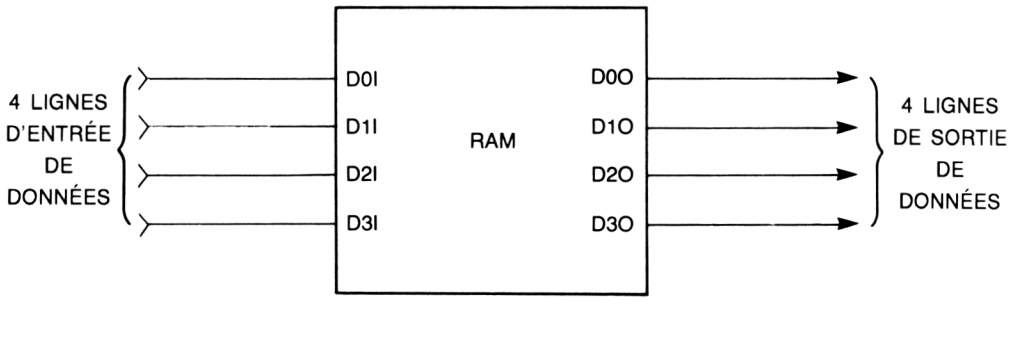


Figure 2.4 : *Schéma d'une RAM ayant des lignes d'entrée et des lignes de sortie séparées.*

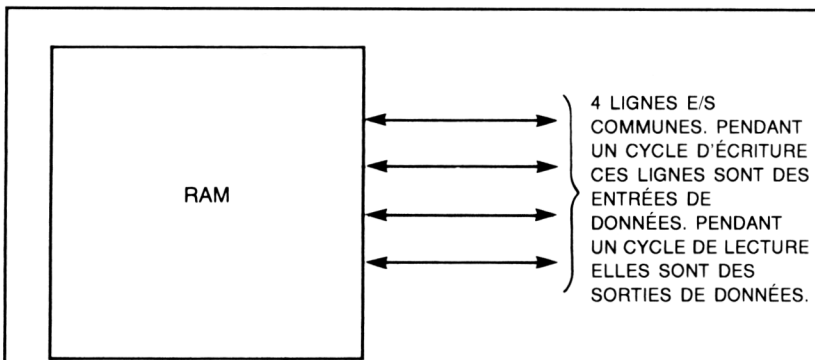


Figure 2.5 : *Schéma d'une RAM ayant des lignes d'entrée et des lignes de sortie communes.*

lignes de données d'*entrée* et de *sortie* de la RAM. Celle-ci peut avoir des entrées-sorties soit séparées, soit communes. *Entrées-sorties séparées* signifie que la RAM a des lignes d'entrée *et* de sortie de données qui sont distinctes (Voir Figure 2.4). ***Entrées-sorties communes*** signifie que les lignes d'entrée et de sortie de données vont sur les mêmes broches du boîtier (Voir Figure 2.5).

2114A 1024 X 4 BIT STATIC RAM

	2114AL-1	2114AL-2	2114AL-3	2114AL-4	2114A-4	2114A-5
Max. Access Time (ns)	100	120	150	200	200	250
Max. Current (mA)	40	40	40	40	70	70

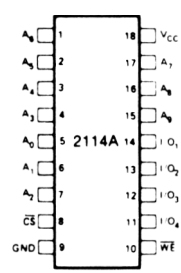
- **HMOS Technology**
- **Low Power, High Speed**
- **Identical Cycle and Access Times**
- **Single +5V Supply ±10%**
- **High Density 18 Pin Package**
- **Completely Static Memory - No Clock or Timing Strobe Required**
- **Directly TTL Compatible: All Inputs and Outputs**
- **Common Data Input and Output Using Three-State Outputs**
- **2114 Upgrade**

The Intel® 2114A is a 4096-bit static Random Access Memory organized as 1024 words by 4-bits using HMOS, a high performance MOS technology. It uses fully DC stable (static) circuitry throughout, in both the array and the decoding, therefore it requires no clocks or refreshing to operate. Data access is particularly simple since address setup times are not required. The data is read out nondestructively and has the same polarity as the input data. Common input/output pins are provided.

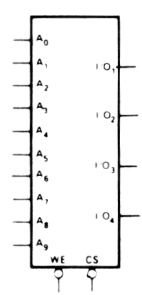
The 2114A is designed for memory applications where the high performance and high reliability of HMOS, low cost, large bit storage, and simple interfacing are important design objectives. The 2114A is placed in an 18-pin package for the highest possible density.

It is directly TTL compatible in all respects: inputs, outputs, and a single +5V supply. A separate Chip Select ($\overline{CS}$) lead allows easy selection of an individual package when outputs are or-tied.

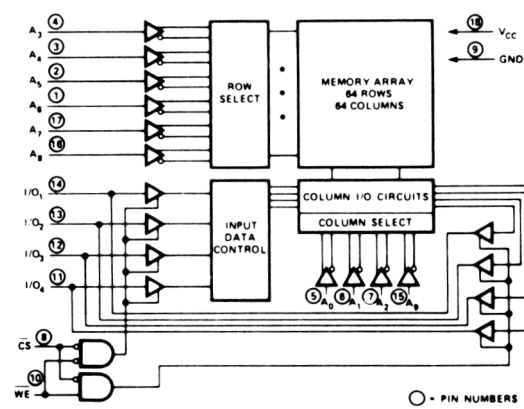
PIN CONFIGURATION



LOGIC SYMBOL



BLOCK DIAGRAM



PIN NAMES

$A_0 - A_9$	ADDRESS INPUTS	V_{CC}	POWER (+5V)
$\overline{WE}$	WRITE ENABLE	GND	GROUND
$\overline{CS}$	CHIP SELECT		
$I/O_1 - I/O_4$	DATA INPUT/OUTPUT		

Figure 2.6 : *Vue partielle d'une feuille de spécifications et schéma de la RAM statique 2114 1K × 4 à entrées-sorties communes.*

Comme une RAM ne peut faire l'objet que d'une opération d'écriture ou de lecture, seules les lignes d'entrée et de sortie de données sont nécessaires. De plus, ces deux fonctions s'excluent mutuellement : on ne peut jamais lire et écrire en mémoire en même temps. Les lignes d'entrée et celles de sortie ne sont donc jamais utilisées en même temps. Ceci permet de réduire le nombre des broches nécessaires aux fonctions de lecture et d'écriture de la RAM. Ceci permet aussi des lignes d'entrée-sortie communes. En outre, les lignes d'entrée-sortie de données peuvent être *multiplexées dans le temps*. Cela signifie que pendant une opération de lecture en mémoire, les lignes d'entrée-sortie de données servent aux sorties

A.C. CHARACTERISTICS T_A = 0°C to 70°C V_{CC} = 5V ± 10%, unless otherwise noted

READ CYCLE [1]

SYMBOL	PARAMETER	2114AL-1 Min.	2114AL-1 Max.	2114AL-2 Min.	2114AL-2 Max.	2114AL-3 Min.	2114AL-3 Max.	2114A-4/L-4 Min.	2114A-4/L-4 Max.	2114A-5 Min.	2114A-5 Max.	UNIT
t_{RC}	Read Cycle Time	100		120		150		200		250		ns
t_A	Access Time		100		120		150		200		250	ns
t_{CO}	Chip Selection to Output Valid		70		70		70		70		85	ns
t_{CX}	Chip Selection to Output Active	10		10		10		10		10		ns
t_{OTD}	Output 3-state from Deselection		30		35		40		50		60	ns
t_{OHA}	Output Hold from Address Change	15		15		15		15		15		ns

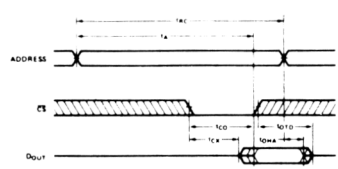
WRITE CYCLE [2]

SYMBOL	PARAMETER	2114AL-1 Min.	2114AL-1 Max.	2114AL-2 Min.	2114AL-2 Max.	2114AL-3 Min.	2114AL-3 Max.	2114A-4/L-4 Min.	2114A-4/L-4 Max.	2114A-5 Min.	2114A-5 Max.	UNIT
t_{WC}	Write Cycle Time	100		120		150		200		250		ns
t_W	Write Time	75		75		90		120		135		ns
t_{WR}	Write Release Time	0		0		0		0		0		ns
t_{OTW}	Output 3-state from Write		30		35		40		50		60	ns
t_{DW}	Data to Write Time Overlap	70		70		90		120		135		ns
t_{DH}	Data Hold from Write Time	0		0		0		0		0		ns

NOTES
1. A Read occurs during the overlap of a low $\overline{CS}$ and a high $\overline{WE}$.
2. A Write occurs during the overlap of a low $\overline{CS}$ and a low $\overline{WE}$. t_W is measured from the latter of $\overline{CS}$ or $\overline{WE}$ going low to the earlier of $\overline{CS}$ or $\overline{WE}$ going high.

WAVEFORMS

READ CYCLE ③



NOTES
3. $\overline{WE}$ is high for a Read Cycle.
4. If the $\overline{CS}$ low transition occurs simultaneously with the $\overline{WE}$ low transition the output buffers remain in a high impedance state.
5. $\overline{WE}$ must be high during all address transitions.

WRITE CYCLE

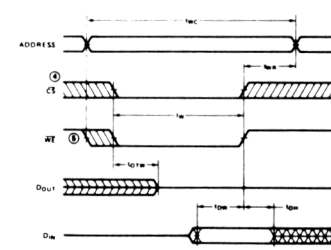


Figure 2.7 : *Extrait de feuilles de caractéristiques montrant les paramètres des temps d'une mémoire 2114.*

de données et que pendant une opération d'écriture, elles servent aux entrées. L'état logique de la ligne de commande R/W détermine à tout moment la fonction des broches d'entrée-sortie. La Figure 2.6 montre le brochage et le circuit synoptique d'une mémoire 2114.

Examinons maintenant les principaux paramètres de cette mémoire. (*Note :* Les informations données ici s'appliquent à la plupart des mémoires à semi-conducteur, même si elles ne comportent pas d'entrées-sorties communes.)

Dans cet exemple, nous considérerons le 2114 au niveau utilisateur seulement. De plus, nous supposerons que le temps d'accès de la mémoire est compatible avec la vitesse globale du système ; nous supposerons aussi que nous n'avons pas besoin de produire des temps d'attente pour des mémoires lentes.

La Figure 2.7 donne une vue partielle d'une feuille de caractéristiques et présente des diagrammes de temps relatifs aux communications électriques avec une mémoire 2114. Considérons d'abord l'écriture de données en mémoire. Ces opérations suivent la séquence générale décrite dans la section 2.3 de ce chapitre. Nous allons cependant décrire le processus d'une façon plus détaillée et introduire les spécifications réelles de la mémoire dont nous nous occupons.

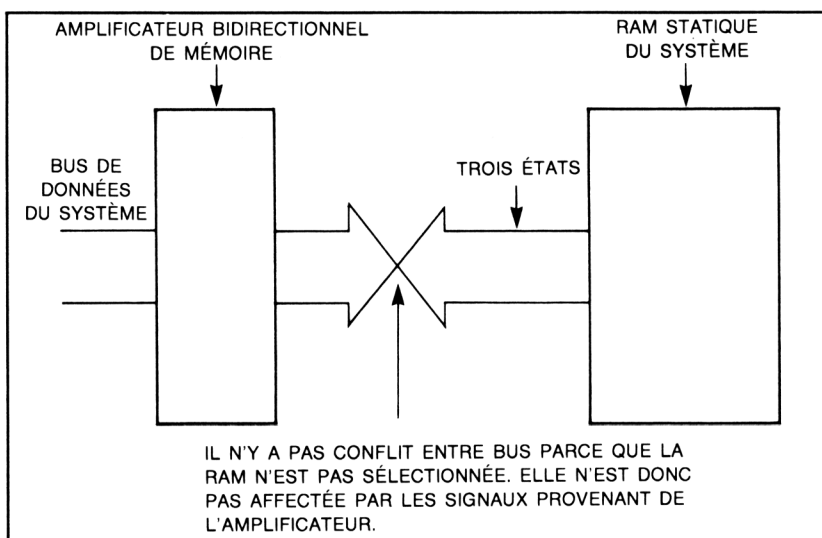


Figure 2.8 : *Schéma montrant comment un conflit peut exister entre lignes de sorties de données et sorties des amplificateurs de données d'une mémoire 2114. On peut l'éviter par une commande appropriée d'une ligne de commande de direction des amplificateurs de données de la mémoire.*

2.5 - Déroulement des opérations d'écriture de données dans une 2114

1. Les valeurs d'adresse sont appliquées à l'adresse où les données doivent être écrites.

2. Les données sont appliquées aux lignes d'entrée-sortie de données de la mémoire. Ces lignes sont à trois états, sinon il y aurait conflit entre les données appliquées et celles stockées dans la mémoire ; autrement dit, elles tenteraient toutes de commander les mêmes lignes d'entrée-sortie. Ceci apparaît à la Figure 2.8.

3. Les entrées $\overline{WE}$ et $\overline{CS}$ de la mémoire sont ensuite activées à peu près au même instant. Quand l'entrée $\overline{WE}$ est au 0 logique, les lignes d'entrée-sortie agissent en entrées. La mémoire ne devrait donc pas être sélectionnée avant activation de l'entrée $\overline{WE}$. Ceci apparaît à la Figure 2.9.

Maintenant que nous sommes au courant de ce qui se passe chaque fois que le CPU écrit des données dans une mémoire 2114, passons à l'examen de ce qui se produit lorsque cette mémoire fait l'objet d'une lecture.

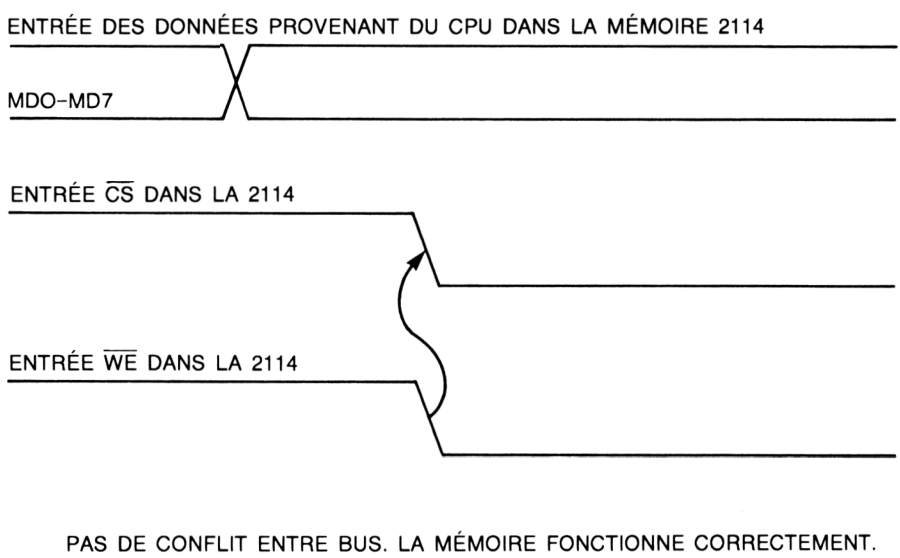


Figure 2.9 : *Diagramme des temps montrant la façon correcte de commander les amplificateurs de données de la mémoire. Dans ce cas, on évite le conflit montré en Figure 2.8.*

2.6 - Déroulement des opérations de lecture de données d'une 2114

1. L'adresse est envoyée à la mémoire où les données doivent être lues.

2. La commande $\overline{CS}$ est ensuite établie. A ce moment, la ligne d'entrée $\overline{WE}$ est au 1 logique. Les lignes de données d'entrée-sortie de la mémoire 2114 sont valides.

3. Les lignes de données d'entrée-sortie émettent les données qui ont été précédemment stockées dans l'emplacement d'adresse.

Nous répétons que la lecture de données d'une RAM statique s'opère de la même façon que celle d'une ROM. Ceci veut dire que les mêmes règles de temps et d'amplification s'appliquent.

Maintenant que nous avons compris comment fonctionne la RAM 2114, connectons-la au microprocesseur Z80.

2.7 - Connexion des lignes d'adresses au Z80

Pour illustrer comment le Z80 communique avec une RAM, nous allons maintenant présenter l'étude complète d'un système général à RAM. En examinant ce système, nous étudierons les problèmes pouvant survenir lors de l'utilisation de RAM avec le Z80. Bien qu'il s'agisse de renseignements d'ordre général, ils sont importants pour tous ceux qui veulent employer des RAM statiques avec un Z80.

On peut voir comment le bus d'adresses du Z80 est relié à la RAM, en examinant le plan d'adressage du système. La Figure 2.10 en montre une partie. Un plan d'adressage permet de déterminer où placer la RAM dans l'espace d'adressage qui peut comporter jusqu'à 64 K adresses. Le plan de la Figure 2.10 montre l'espace mémoire de la RAM aux emplacements 3000 à 3FFF en hexadécimal. Cet espace d'adressage donne au système 4096 ou 4 K × 8 emplacements de mémoire RAM.

Dans cet exemple, nous allons installer une mémoire RAM 4 K × 8 qui communique avec le Z80. Si nous examinons la feuille de caractéristiques de la Figure 2.6, nous pouvons voir que chaque mémoire a 4 lignes d'entrée-sortie. Comme le microprocesseur Z80 communique au moyen de huit lignes de données en parallèle - c'est-à-dire un octet à la fois - on doit employer deux mémoires 2114 en parallèle pour former un mot complet de 8 bits.

La première chose à faire est donc de déterminer le nombre de mémoires nécessaires à l'établissement de l'espace RAM total. Ceci peut être calculé

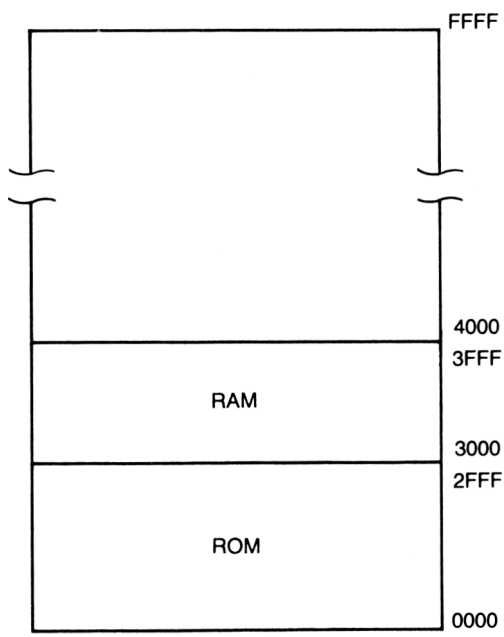


Figure 2.10 : *Plan d'adressage partiel montrant l'espace d'adressage d'une ROM et d'une RAM.*

de la façon suivante : l'espace RAM existant vaut 4096 octets ; chaque mémoire 2114 stocke jusqu'à 1024 quartets ; donc pour chacun des 1024 octets en mémoire 2114, il faut deux boîtiers. Il nous faut donc huit mémoires 2114 pour l'espace RAM complet. Nous n'avons évidemment pas besoin d'utiliser la totalité de l'espace disponible.

Dans cet exemple, notre système à Z80 ne requiert que 1 K octets de RAM. Nous supposerons donc que le système a été prévu avec 4 K octets d'espace mémoire. (*Note :* C'est généralement une bonne idée de prévoir dans votre projet plus d'espace mémoire que nécessaire, si votre système le permet. Ceci vous donnera la possibilité, au besoin, d'accroître ultérieurement la mémoire. Gardez présent à l'esprit qu'ajouter de la mémoire dans un système qui n'a pas été prévu pour cela, risque d'être une opération difficile.)

Nous savons maintenant que notre système demande huit mémoires 2114 pour réaliser l'espace mémoire complet. Chaque 2114 a 10 lignes d'adres-

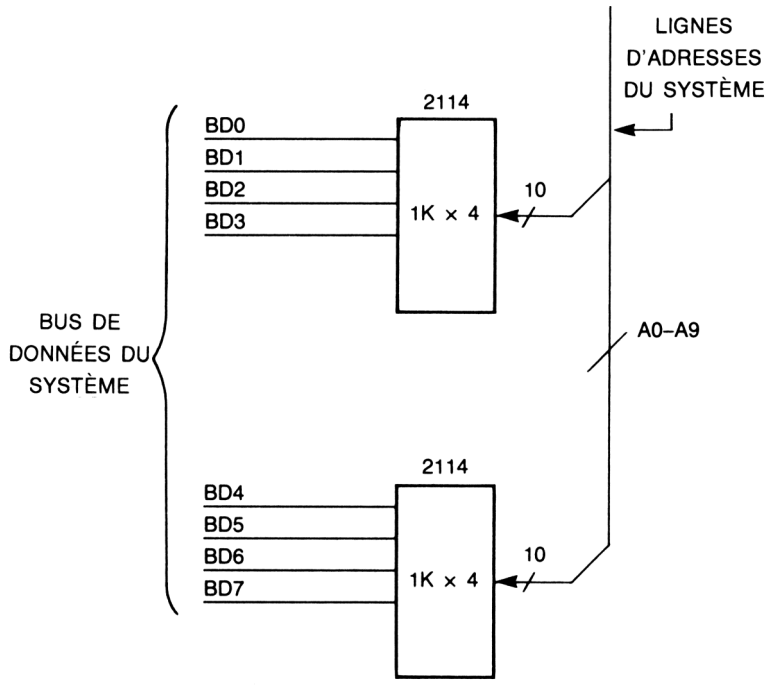


Figure 2.11 : *On utilise deux mémoires 2114 en parallèle pour établir un système mémoire complet à 8 bits.*

ses, A0 à A9, et toutes ces lignes doivent être reliées en parallèle. Ceci veut dire que les lignes A0 des huit boîtiers doivent être reliées ensemble, de même pour les lignes A1 et ainsi de suite. (Voir Figure 2.12.)

Les autres lignes d'adresses, A10 à A15, servent à décoder l'espace d'adressage RAM pour le distinguer du reste de l'espace d'adressage. On utilise deux niveaux de décodage. Les lignes d'adresses A10 et A11 servent à choisir la paire de mémoires 2114 appropriée ; les lignes d'adresses A11 à A15 servent à choisir l'espace C000-D000 dans tout l'espace d'adressage existant.

On peut utiliser plusieurs techniques différentes de décodage d'adresse pour une application donnée. Le circuit de la Figure 2.13 correspond à l'une d'elle. Il vous donnera une idée générale de la méthode permettant d'effectuer une sélection logique de mémoire. Bien que ces circuits fonctionnent, ils ne constituent pas une solution définitive. Toutes les

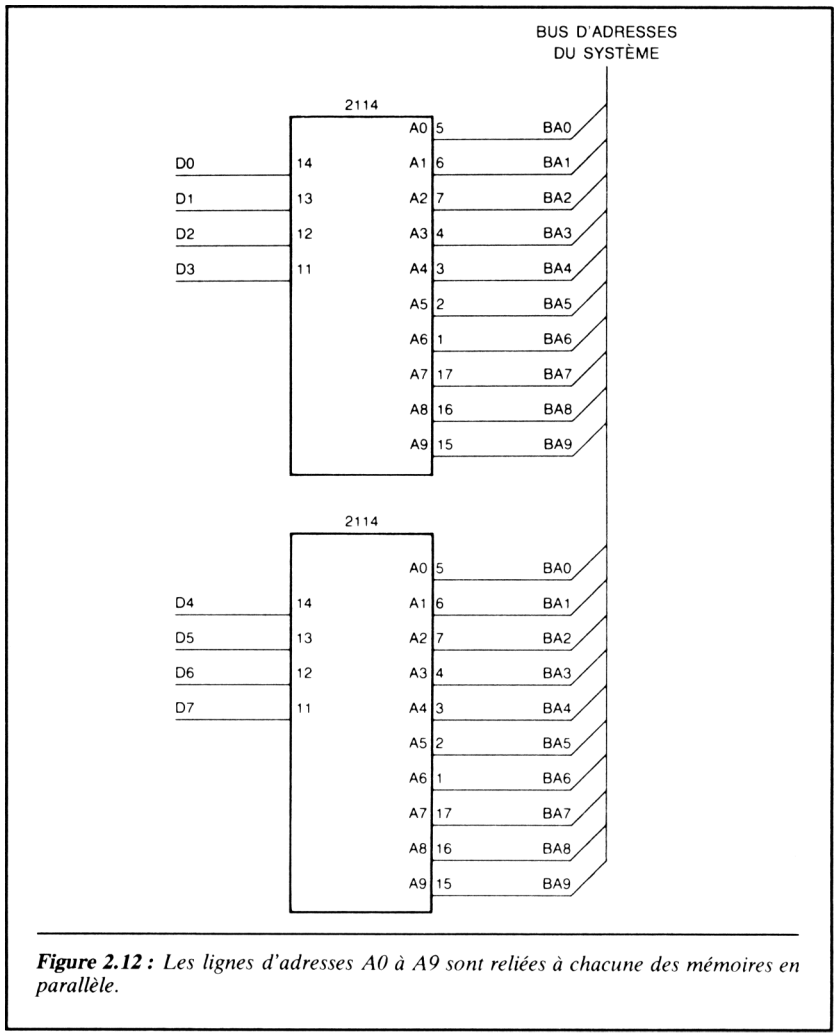


Figure 2.12 : *Les lignes d'adresses A0 à A9 sont reliées à chacune des mémoires en parallèle.*

solutions doivent être adaptées à l'application que l'on veut faire et dépendent normalement de considérations de vitesse, de charge de bus et de plan d'adressage. La Figure 2.13 montre les circuits requis pour réaliser le décodage à deux niveaux de l'espace RAM de cet exemple.

Notez ici que l'on peut, ou non, utiliser des amplificateurs d'adresses. Rappelez-vous que nous avons expliqué l'emploi d'amplificateurs d'adresses au Chapitre 1. Les mêmes règles de charge et de vitesse s'appliquent aussi bien aux RAM qu'aux ROM.

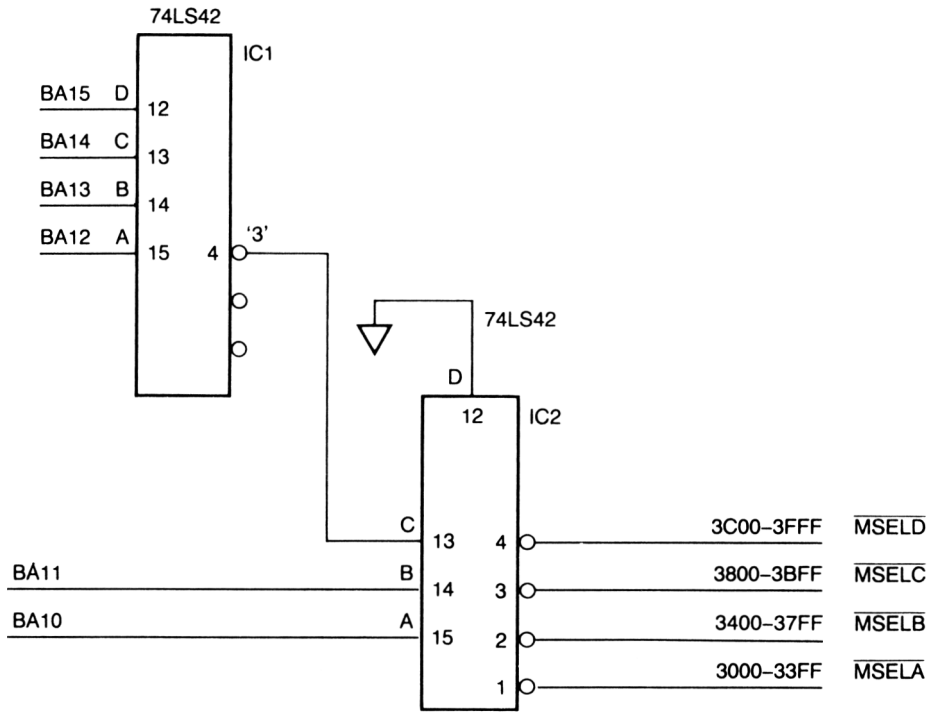


Figure 2.13 : *Schéma montrant les circuits de décodage formant la logique de sélection de mémoire.*

2.8 - Connexion des lignes de données - Sans amplification

Nous allons maintenant vous montrer comment connecter les sorties de données de la mémoire au bus de données du Z80. Dans cet exemple, nous n'emploierons pas d'amplificateur de données ; nous supposerons que la charge des lignes de bus de données du Z80 n'excède pas les spécifications concernant les lignes d'entrée-sortie de données de la mémoire. Nous connecterons donc simplement les sorties de la mémoire 2114 en parallèle, puis les huit lignes d'entrée-sortie résultantes, au bus de données du Z80. La connexion est simple. Le processus est montré en Figure 2.14.

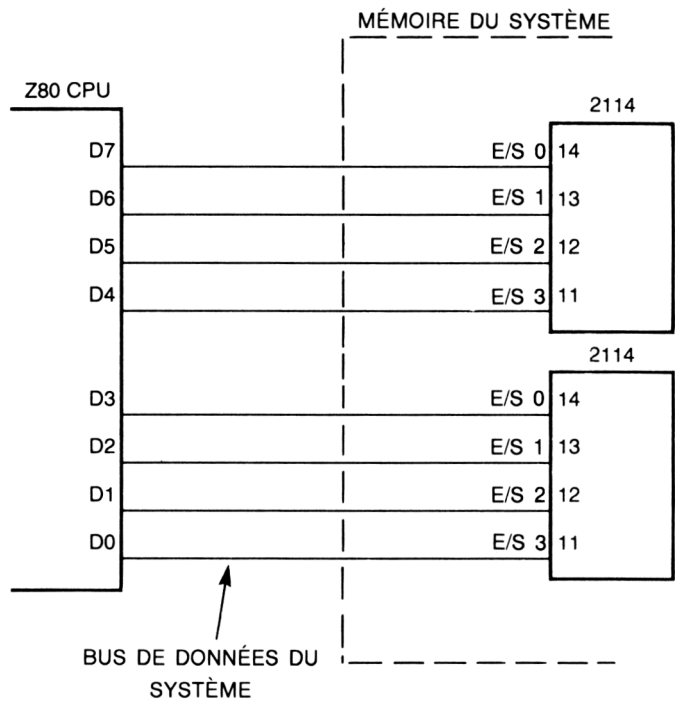


Figure 2.14 : *Schéma montrant les liaisons matérielles des lignes de données entre 2114 et microprocesseur Z80.*

Il est important de se rappeler que les lignes d'entrée-sortie d'un système de mémoire 2114 sont commandées par les entrées $\overline{WE}$ et $\overline{CS}$. Si votre application ne nécessite pas d'amplificateur de données, une liaison simple entre lignes de sortie de mémoire et lignes du bus de données du Z80 suffit.

2.9 - Production des commandes de lecture et d'écriture

Quand le microprocesseur Z80 communique avec la mémoire, trois lignes de sorties, $\overline{MERQ}$, $\overline{RD}$ et $\overline{WR}$, servent à produire les commandes référencées $\overline{MEMR}$ et $\overline{MEMW}$. La Figure 2.15 montre un circuit logique

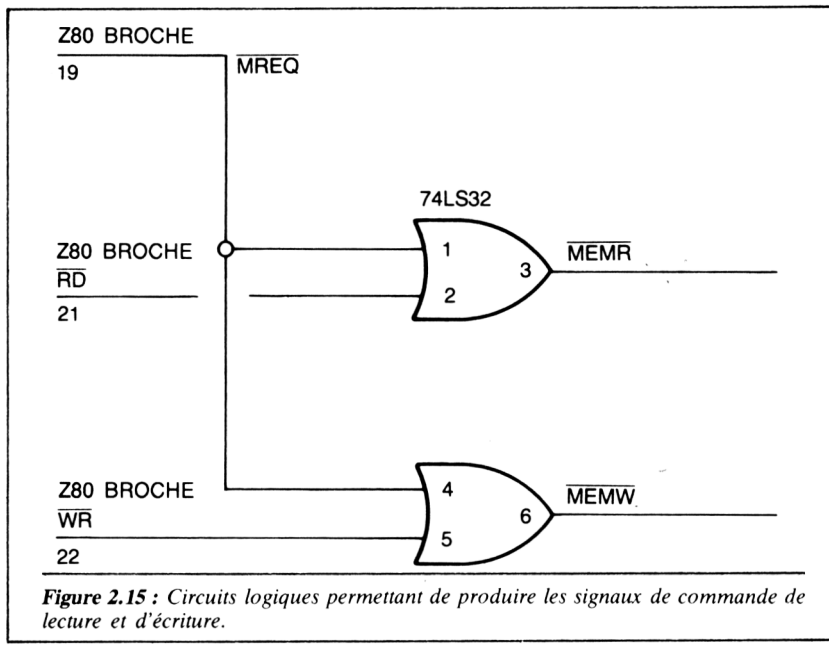


Figure 2.15 : *Circuits logiques permettant de produire les signaux de commande de lecture et d'écriture.*

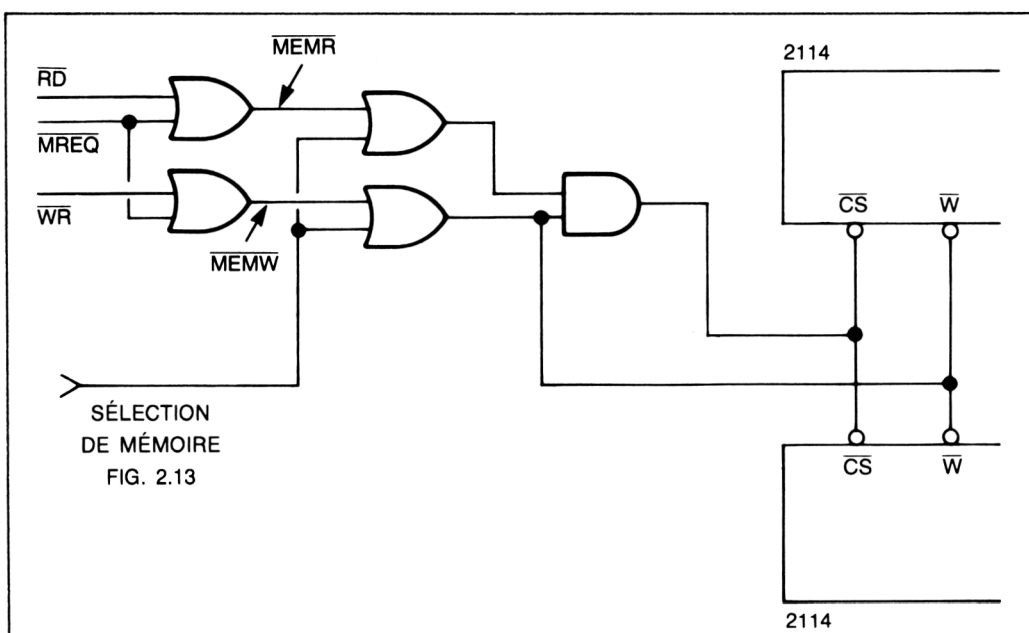


Figure 2.16 : *Schéma utilisé pour activer les entrées $\overline{CS}$ et $\overline{WE}$ des mémoires 2114.*

classique qui produit un signal de lecture et d'écriture à partir de l'état logique de ces trois lignes de sortie du Z80.

Nous allons maintenant employer les lignes de sortie visibles en Figure 2.15 avec les circuits de décodage d'adresses de la Figure 2.13 pour commander les entrées $\overline{WE}$ et $\overline{CS}$ des RAM. (*Note :* Rappelez-vous que ces renseignements sont généraux et ne s'appliquent *pas seulement* aux mémoires 2114, mais à la plupart des mémoires statiques. La mémoire 2114 sert simplement ici d'exemple.)

La Figure 2.16 montre comment les lignes $\overline{MEMR}$ et $\overline{MEMW}$ sont connectées aux entrées $\overline{WE}$ et $\overline{CS}$ de la mémoire 2114. Ce circuit active les

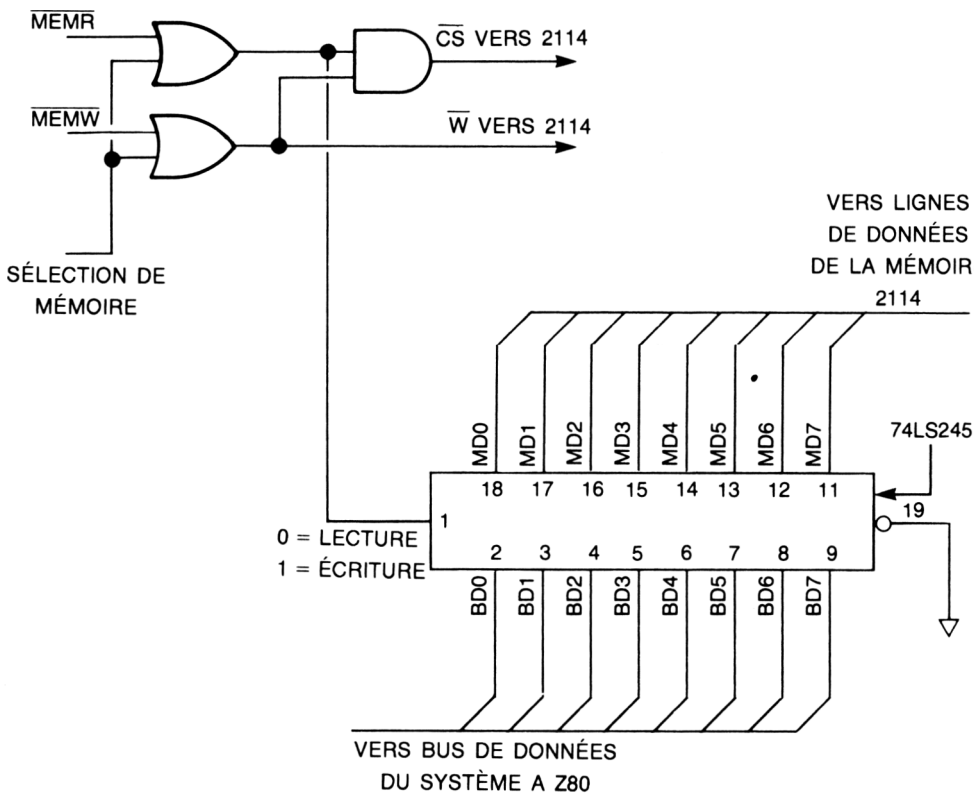


Figure 2.17 : *Schéma partiel d'emploi d'amplificateur de données pour mémoire 2114. Notez qu'un amplificateur bidirectionnel 74LS245 est employé. Ceci est dû aux caractéristiques d'entrée-sortie communes de la mémoire.*

entrées $\overline{WE}$ et $\overline{CS}$ de la mémoire à peu près à l'instant où survient un ordre d'écriture. Cependant, durant l'opération de lecture, seule l'entrée $\overline{CS}$ de la mémoire 2114 sera activée.

2.10 - Utilisation de lignes de données amplifiées avec des RAM statiques

Dans la partie précédente, nous avons relié les lignes d'entrée-sortie de la mémoire 2114 directement aux lignes de données du Z80. Cette liaison est électriquement valide si, dans son ensemble, on n'a pas besoin d'amplifier les données sur le bus de données du Z80. Nous allons maintenant supposer que votre application exige un amplificateur *bidirectionnel* de données sur les lignes du bus de données du Z80. Dans ce cas, un amplificateur de données doit être installé sur les lignes de sortie de la mémoire. Ceci est dû au fait que, durant une opération de lecture, les données issues de la mémoire doivent être amplifiées sur le bus de données.

La Figure 2.17 montre un moyen pour installer des amplificateurs de données dans un système à RAM statique. Notez que les lignes d'entrée-sortie de la mémoire 2114 sont toutes connectées comme avant ; cependant elles ne sont pas connectées directement aux lignes de données du Z80. Au lieu de cela, elles sont connectées à un côté d'un amplificateur bidirectionnel. La commande de direction de l'amplificateur de données dépend de l'état logique du signal $\overline{MEMR}$. Chaque fois que ce signal est un 0 logique et que l'espace mémoire est validé, les amplificateurs amplifient les données provenant de la RAM et allant vers le bus de données du système.

2.11 - RAM statique complète 4 K × 8 bits

La Figure 2.18 montre un système de RAM statique complet communiquant avec le microprocesseur Z80. Remarquez que ce système comporte un amplificateur de données. Si votre application n'en nécessite pas, vous pouvez le supprimer en connectant simplement les lignes MD directement aux lignes du bus de données du système.

2.12 - Autre RAM statique : la 6116

Nous allons présenter maintenant une autre RAM statique classique qui peut être facilement employée dans un système commandé par Z80. Il s'agit

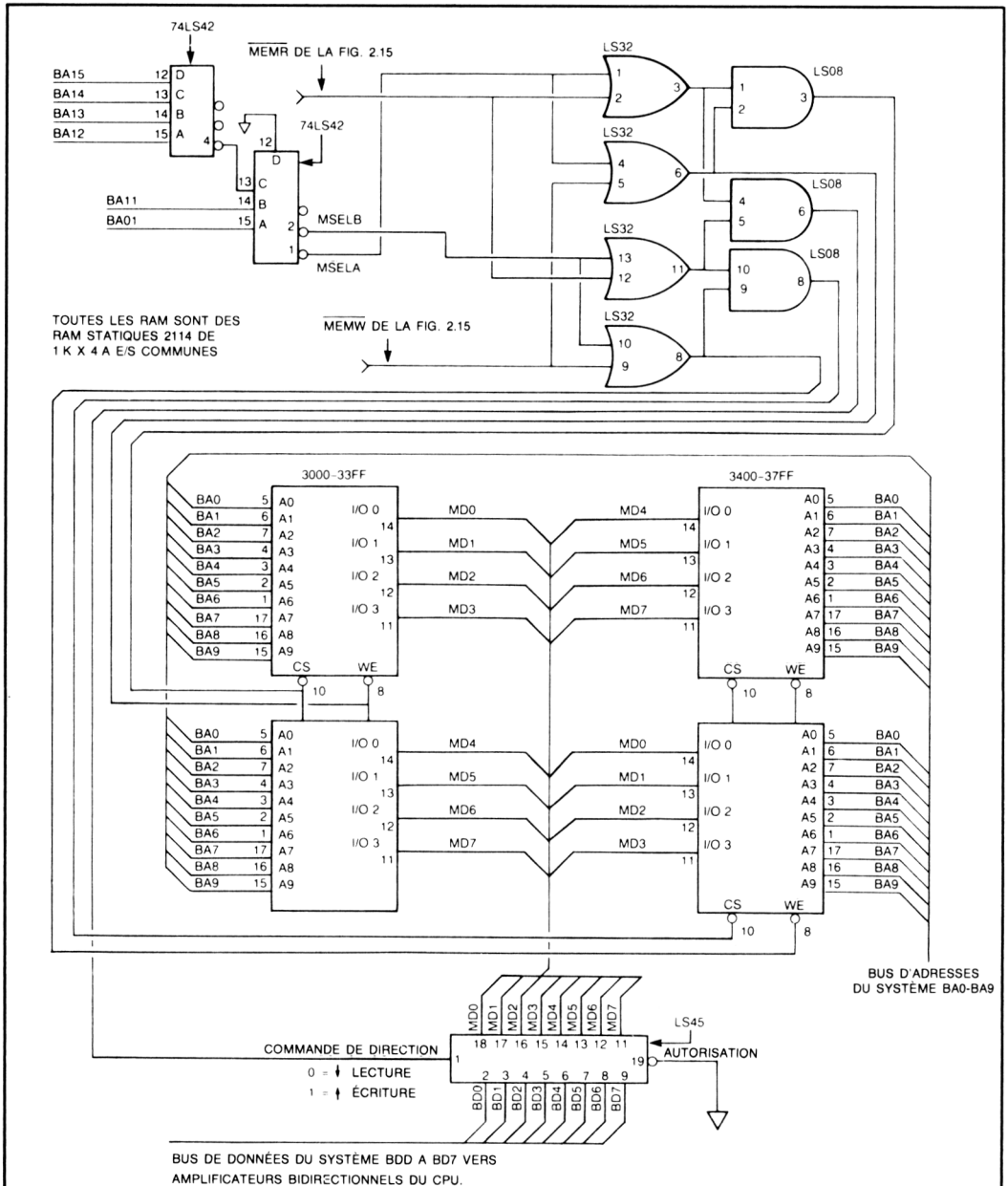


Figure 2.18 : *Schéma complet de système de mémoire 2K × 8 bits pour microprocesseur Z80.*

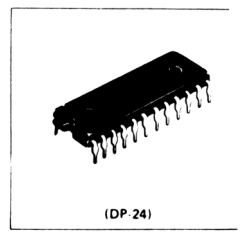
de la 6116, RAM statique, à entrées-sorties communes, 2 K × 8. La Figure 2.19 montre une partie de la feuille de caractéristiques de ce composant. La 6116 a des caractéristiques de fonctionnement similaires à celles de la RAM statique 2114.

HM6116P-2, HM6116P-3, HM6116P-4

2048-word × 8-bit High Speed Static CMOS RAM

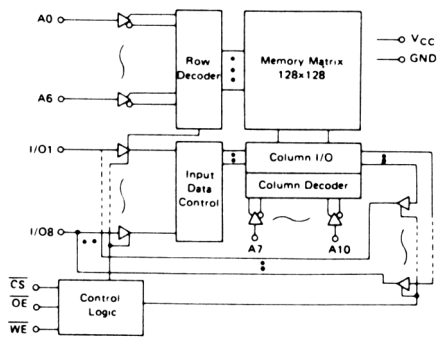
■ FEATURES

- Single 5V Supply and High Density 24 pin Package
- High Speed: Fast Access Time 120ns/150ns/200ns (max.)
- Low Power Standby and Low Power Operation; Standby: 100μW (typ.) Operation: 180mW (typ.)
- Completely Static RAM: No clock or Timing Strobe Required
- Directly TTL Compatible: All Input and Output
- Pin Out Compatible with Standard 16K EPROM/MASK ROM
- Equal Access and Cycle Time

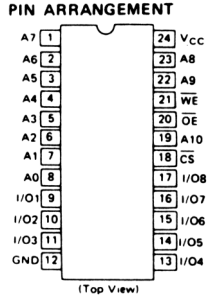


(DP-24)

■ FUNCTIONAL BLOCK DIAGRAM



■ PIN ARRANGEMENT



■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Rating	Unit
Voltage on Any Pin Relative to GND	V_{IN}	−0.5 to +7.0	V
Operating Temperature	T_{opr}	0 to +70	°C
Storage Temperature	T_{stg}	−55 to +125	°C
Temperature Under Bias	T_{bias}	−10 to +85	°C
Power Dissipation	P_T	1.0	W

■ TRUTH TABLE

$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	Mode	V_{CC} Current	I/O Pin	Ref. Cycle
H	X	X	Not Selected	I_{SB}, I_{SB1}	High Z	
L	L	H	Read	I_{CC}	D_{out}	Read Cycle (1) ~ (3)
L	H	L	Write	I_{CC}	D_{in}	Write Cycle (1)
L	L	L	Write	I_{CC}	D_{in}	Write Cycle (2)

Figure 2.19 : *Feuille de données de la RAM statique à entrées/sorties communes,*

■ RECOMMENDED DC OPERATING CONDITIONS (T_a = 0 to +70°C)

Item	Symbol	min	typ	max	Unit
Supply Voltage	V_{CC}	4.5	5.0	5.5	V
	GND	0	0	0	V
Input Voltage	V_{IH}	2.2	3.5	6.0	V
	V_{IL}	−1.0*	–	0.8	V

* Pulse Width: 50 ns, DC: V_{IL} min = −0.3V

■ DC AND OPERATING CHARACTERISTICS (V_{CC} = 5V ±10%, GND = 0V, T_a = 0 to +70°C)

Item	Symbol	Test Conditions	HM6116P-2			HM6116P-3/-4			Unit
			min	typ*	max	min	typ*	max	
Input Leakage Current	$\|I_{LI}\|$	V_{CC} = 5.5V, V_{in} = GND to V_{CC}	–	–	10	–	–	10	μA
Output Leakage Current	$\|I_{LO}\|$	$\overline{CS}$ = V_{IH} or $\overline{OE}$ = V_{IH}, $V_{I/O}$ = GND to V_{CC}	–	–	10	–	–	10	μA
Operating Power Supply Current	I_{CC}	$\overline{CS}$ = V_{IL}, $I_{I/O}$ = 0mA	–	40	80	–	35	70	mA
	I_{CC1}**	V_{IH} = 3.5V, V_{IL} = 0.6V, $I_{I/O}$ = 0mA	–	35	–	–	30	–	mA
Average Operating Current	I_{CC2}	Min. cycle, duty = 100%	–	40	80	–	35	70	mA
Standby Power Supply Current	I_{SB}	$\overline{CS}$ = V_{IH}	–	5	15	–	5	15	mA
	I_{SB1}	$\overline{CS} \geq V_{CC} - 0.2V$, $V_{in} \geq V_{CC} - 0.2V$ or $V_{in} \leq 0.2V$	–	0.02	2	–	0.02	2	mA
Output Voltage	V_{OL}	I_{OL} = 4mA	–	–	0.4	–	–	–	V
		I_{OL} = 2.1mA	–	–	–	–	–	0.4	
	V_{OH}	I_{OH} = −1.0mA	2.4	–	–	2.4	–	–	V

* V_{CC} = 5V, T_a = 25°C
** Reference Only

■ AC CHARACTERISTICS (V_{CC} = 5V ±10%, T_a = 0 to +70°C)

● AC TEST CONDITIONS

Input Pulse Levels: 0.8 to 2.4V
Input Rise and Fall Times: 10 ns
Input and Output Timing Reference Levels: 1.5V
Output Load: 1TTL Gate and C_L = 100pF (including scope and Jig)

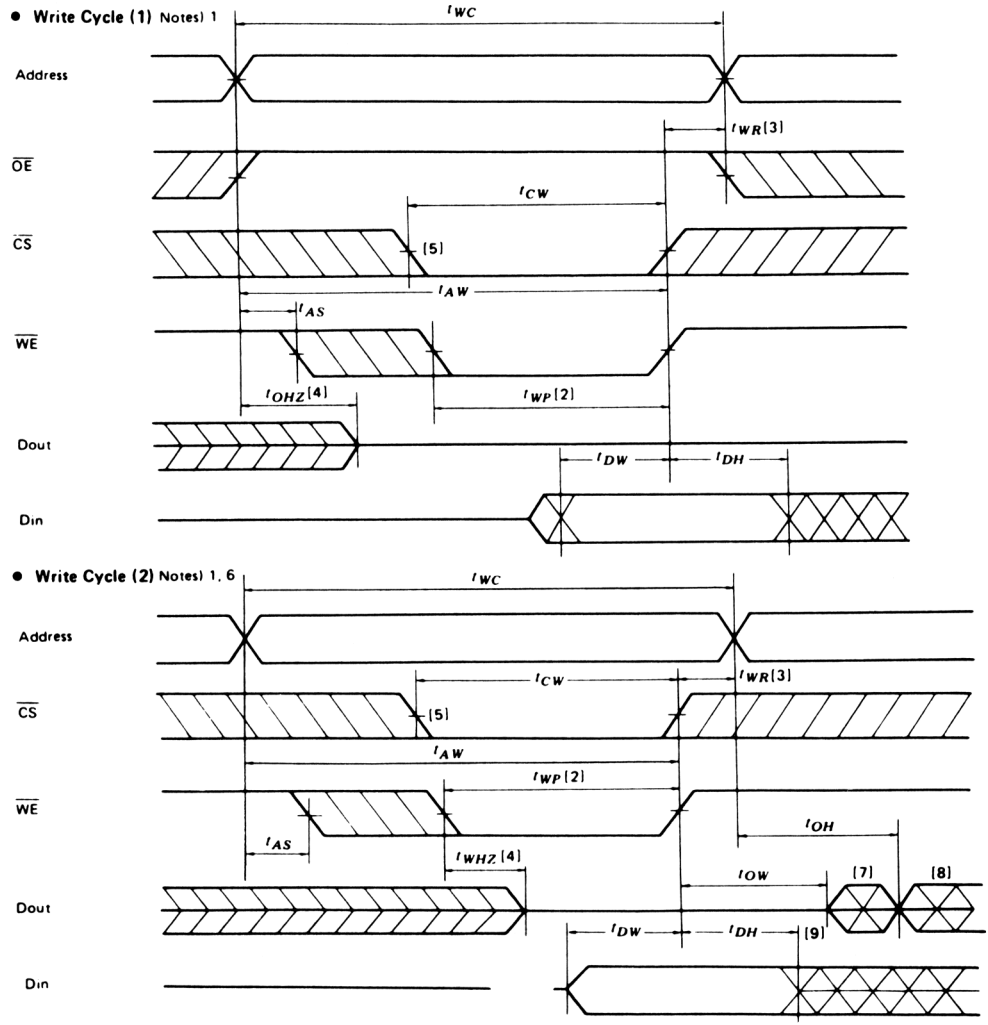
● READ CYCLE

Item	Symbol	HM6116P-2		HM6116P-3		HM6116P-4		Unit
		min	max	min	max	min	max	
Read Cycle Time	t_{RC}	120	–	150	–	200	–	ns
Address Access Time	t_{AA}	–	120	–	150	–	200	ns
Chip Select Access Time	t_{ACS}	–	120	–	150	–	200	ns
Chip Selection to Output in Low Z	t_{CLZ}	10	–	15	–	15	–	ns
Output Enable to Output Valid	t_{OE}	–	80	–	100	–	120	ns
Output Enable to Output in Low Z	t_{OLZ}	10	–	15	–	15	–	ns
Chip deselection to Output in High Z	t_{CHZ}	0	40	0	50	0	60	ns
Chip Disable to Output in High Z	t_{OHZ}	0	40	0	50	0	60	ns
Output Hold from Address Change	t_{OH}	10	–	15	–	15	–	ns

● WRITE CYCLE

Item	Symbol	HM6116P-2		HM6116P-3		HM6116P-4		Unit
		min	typ	min	max	min	max	
Write Cycle Time	t_{WC}	120	–	150	–	200	–	ns
Chip Selection to End of Write	t_{CW}	70	–	90	–	120	–	ns
Address Valid to End of Write	t_{AW}	105	–	120	–	140	–	ns
Address Set Up Time	t_{AS}	20	–	20	–	20	–	ns
Write Pulse Width	t_{WP}	70	–	90	–	120	–	ns
Write Recovery Time	t_{WR}	5	–	10	–	10	–	ns
Output Disable to Output in High Z	t_{OHZ}	0	40	0	50	0	60	ns
Write to Output in High Z	t_{WHZ}	0	50	0	60	0	60	ns
Data to Write Time Overlap	t_{DW}	35	–	40	–	60	–	ns
Data Hold from Write Time	t_{DH}	5	–	10	–	10	–	ns
Output Active from End of Write	t_{OW}	5	–	10	–	10	–	ns

Figure 2.19 : Feuille de données (suite).



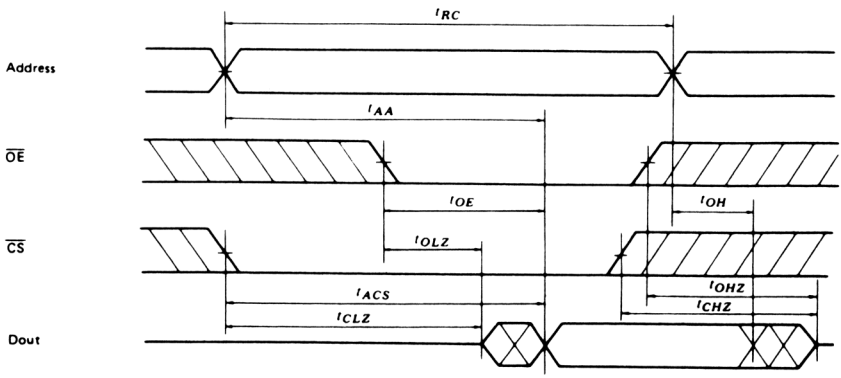
NOTES:
1. $\overline{WE}$ must be high during all address transitions.
2. A write occurs during the overlap (t_{WP}) of a low $\overline{CS}$ and a low $\overline{WE}$.
3. t_{WR} is measured from the earlier of $\overline{CS}$ or $\overline{WE}$ going high to the end of write cycle.
4. During this period, I/O pins are in the output state so that the input signals of opposite phase to the outputs must not be applied.
5. If the $\overline{CS}$ low transition occurs simultaneously with the $\overline{WE}$ low transitions or after the $\overline{WE}$ transition, output remain in a high impedance state.
6. $\overline{OE}$ is continuously low. ($\overline{OE}$ = V_{IL})
7. D_{out} is the same phase of write data of this write cycle.
8. D_{out} is the read data of next address.
9. If $\overline{CS}$ is Low during this period, I/O pins are in the output state. Then the data input signals of opposite phase to the outputs must not be applied to them.

Figure 2.19 : Feuille de données (suite).

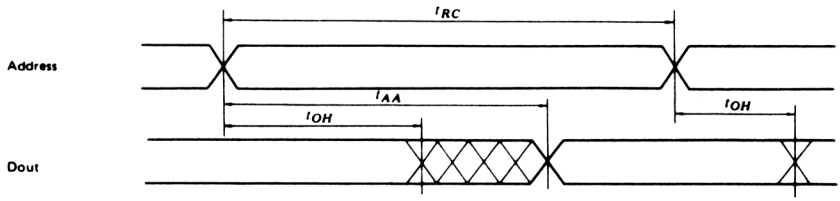
■ **CAPACITANCE** (f = 1MHz, T_a = 25°C)

Item	Symbol	Test Conditions	typ.	max.	Unit
Input Capacitance	C_{in}	V_{in} = 0V	3	5	pF
Input/Output Capacitance	$C_{I/O}$	$V_{I/O}$ = 0V	5	7	pF

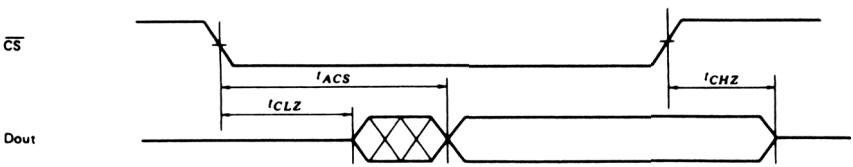
● **Read Cycle (1)** Notes) 1, 5



● **Read Cycle (2)** Notes) 1, 2, 4, 5



● **Read Cycle (3)** Notes) 1, 3, 4, 5



NOTES:
1. $\overline{WE}$ is High for Read Cycle.
2. Device is continuously selected, $\overline{CS}$ = V_{IL}.
3. Address Valid prior to or coincident with $\overline{CS}$ transition Low.
4. $\overline{OE}$ = V_{IL}.
5. When $\overline{CS}$ is Low, the address input must not be in the high impedance state.

Figure 2.19 : Feuille de données (suite).

La Figure 2.20 montre une manière de connecter une RAM 6116 à un système à Z80. A cause de son organisation, un seul boîtier est nécessaire pour une section RAM complète de mémoire. Ceci fait du 6116 un choix idéal, que le système soit petit ou grand.

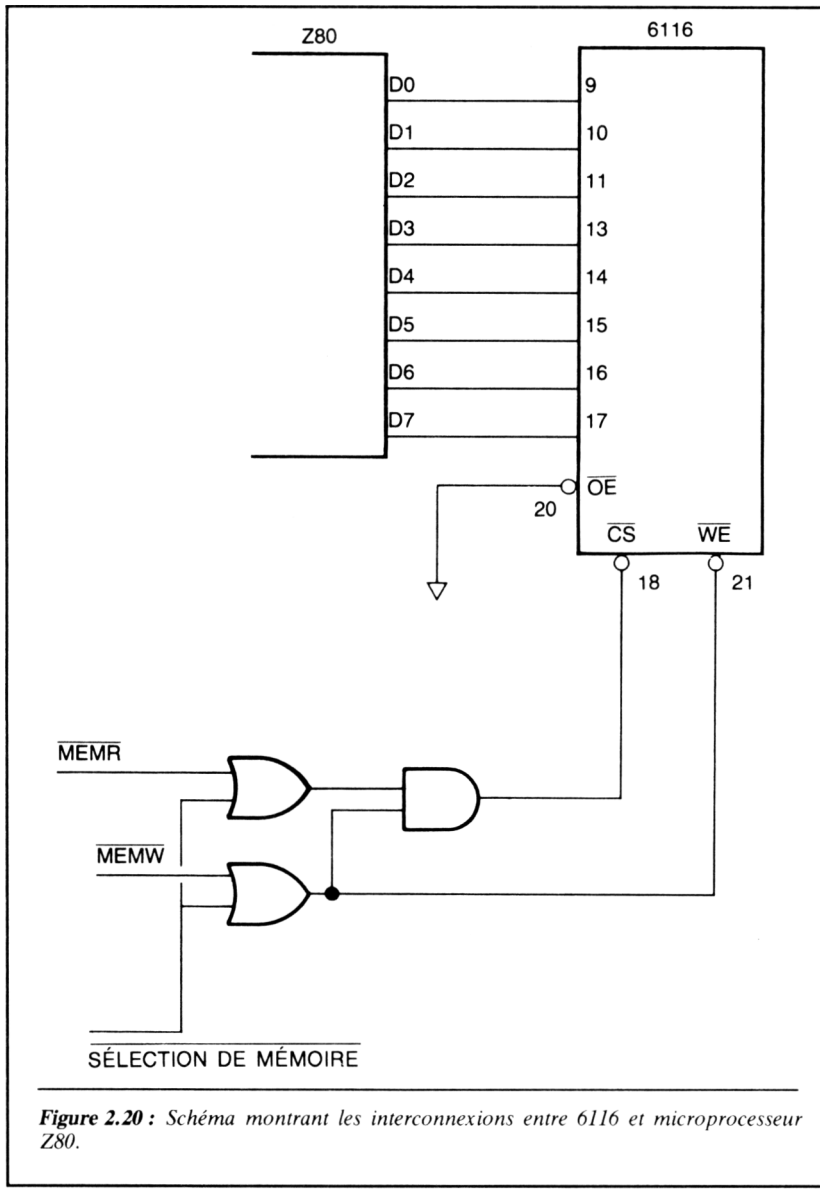


Figure 2.20 : *Schéma montrant les interconnexions entre 6116 et microprocesseur Z80.*

RESUME

Dans ce chapitre, nous avons présenté d'importants renseignements concernant l'utilisation de RAM statiques avec le microprocesseur Z80. Notre étude a commencé par une description générale des séquences de lecture et d'écriture. Nous avons ensuite examiné un système classique à RAM comportant des mémoires 2114, à entrées-sorties communes, de $1\,K \times 4$. D'importants paramètres ont été indiqués et l'on a montré des détails d'opérations de lecture et d'écriture. Nous avons poursuivi en examinant un système de mémoire complet utilisant des mémoires 2114 comme composants. Puis, nous avons réalisé le même système avec une mémoire 6116.

ENTREES ET SORTIES DU Z80

CHAPITRE 3

INTRODUCTION

Les deux opérations matérielles importantes effectuées par un microprocesseur sont la lecture de données provenant d'un dispositif d'entrée et l'écriture de données dans un dispositif de sortie. Nous apprendrons dans ce chapitre comment le Z80 communique avec les dispositifs d'entrées et de sorties. En outre, nous établirons un port standard d'entrées et de sorties (E/S) utilisant des éléments logiques discrets.

Nous avons choisi d'analyser maintenant les opérations d'entrée et de sortie parce que les communications électriques avec les éléments d'entrée-sortie sont similaires à celles des RAM statiques. En lisant ce chapitre, vous constaterez de grandes similitudes dans les communications entre microprocesseur et RAM statiques, et entre microprocesseur et éléments d'entrée-sortie.

Les renseignements donnés dans ce chapitre sont importants. Il faut assimiler les principes des dispositifs d'entrée-sortie pour comprendre ce qui sera exposé dans les chapitres suivants.

3.1 - Aperçu sur les entrées et sorties du Z80

Le Z80 peut provoquer l'entrée ou la sortie de données provenant de, ou dirigées vers un dispositif d'entrée-sortie. Ces deux types d'opérations sont exposés dans les Figures 3.1a et 3.1b. Chaque fois que le Z80 effectue une opération d'entrée-sortie, il se produit une certaine séquence d'opérations physiques dans les circuits du système ; autrement dit, certaines lignes d'adresses, de données et de commande deviennent actives dans un certain ordre. Cette séquence est semblable aux opérations électriques qui ont lieu lors de la lecture et de l'écriture de données en mémoire.

Certaines instructions logicielles provoquent, lorsqu'elles sont exécutées par le Z80, la séquence d'événements correspondant à une opération d'entrée-sortie. (Nous examinerons en détail ces instructions ultérieurement, lorsque nous apprendrons comment interfacer différents dispositifs d'entrée-sortie au Z80.) Mais quelles que soient les instructions logicielles employées, la séquence des opérations se déroulant dans les circuits du système est toujours la même.

Comme le montrent les Figures 3.1a et 3.1b, le Z80 peut recevoir (lire) huit bits de données provenant d'un dispositif d'entrée et envoyer (écrire) huit bits de données vers un dispositif de sortie. Examinons maintenant ceci plus en détail.

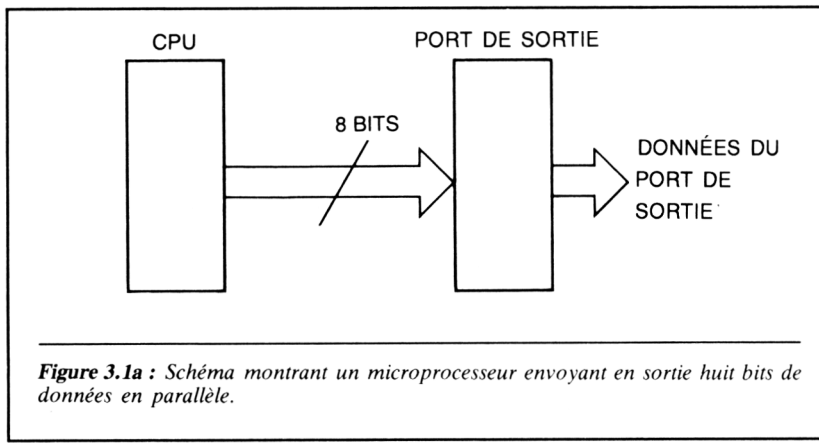


Figure 3.1a : *Schéma montrant un microprocesseur envoyant en sortie huit bits de données en parallèle.*

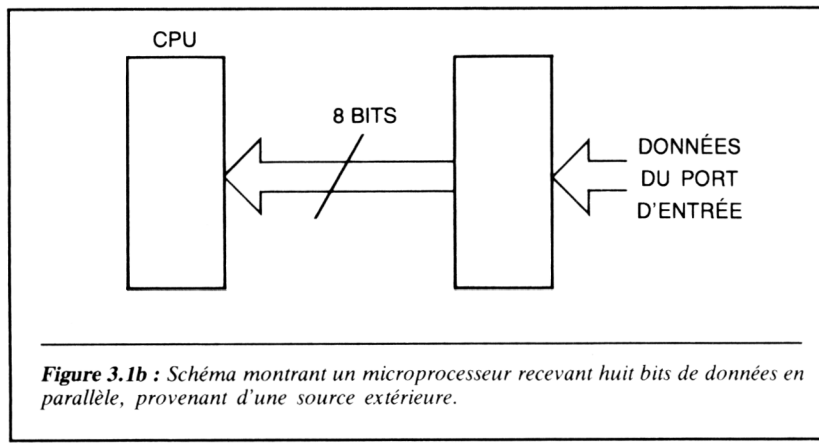


Figure 3.1b : *Schéma montrant un microprocesseur recevant huit bits de données en parallèle, provenant d'une source extérieure.*

3.2 - Adresses de ports

Dans un système à Z80, huit des seize lignes d'adresses servent aux entrées-sorties. Ceci veut dire que le Z80 peut être relié jusqu'à 256 ports d'entrée-sortie à 8 bits. (En supposant que le système utilise la configuration standard d'adressage entrée-sortie et que chaque port ait une adresse distincte.) Si l'on emploie une *sélection d'entrée-sortie linéaire* (on utilise

dans ce cas une ligne d'adresse distincte comme ligne de sélection d'entrée-sortie), il y a huit ports d'entrée-sortie distincts. La sélection linéaire d'entrée-sortie est très utile dans un système ne possédant qu'un petit nombre de dispositifs d'entrée-sortie. (*Note :* Nous nous occuperons dans ce chapitre des entrées-sorties à plan d'adressage. Nous pensons qu'une fois que vous aurez compris ce type de structure d'entrée-sortie, vous pourrez facilement comprendre d'autres types, étant donné qu'ils ne sont qu'une simple variante de ce cas général.)

Rappelons-nous, comme nous l'avons vu au Chapitre 2, que l'adressage des entrées-sorties et des mémoires dans un système à Z80 emploie les mêmes lignes d'adresses. C'est au microprocesseur de séparer électriquement les demandes des mémoires des demandes des entrées-sorties. En outre, nous savons qu'un ensemble de mémoires n'utilise pas toujours toutes les seize lignes d'adresses (A0 à A15). Il en est de même pour le système d'entrée-sortie - souvent, il n'utilise pas toutes les huit lignes d'adresses (A0 à A7). Si, par exemple, il n'y a que cinq ports d'entrée-sortie différents dans le système, il ne faudra que trois lignes d'adresses (puisque trois lignes donneront au total huit combinaisons possibles). Le port d'entrée-sortie standard que nous allons considérer dans ce chapitre, utilise cependant les huit lignes d'adresses pour le décodage des ports.

Rappelons que chaque port d'entrée-sortie d'un système à microprocesseur correspond à une combinaison distincte de huit bits, des lignes d'adresses A0 à A7 du système. La combinaison d'adressage à laquelle va répondre un port est dite *adresse de port.* FF est l'adresse de port pour le dispositif d'entrée-sortie dont nous allons nous occuper dans ce chapitre. La Figure 3.2 donne le schéma d'un circuit qui va détecter cette adresse de port et seulement celle-ci.

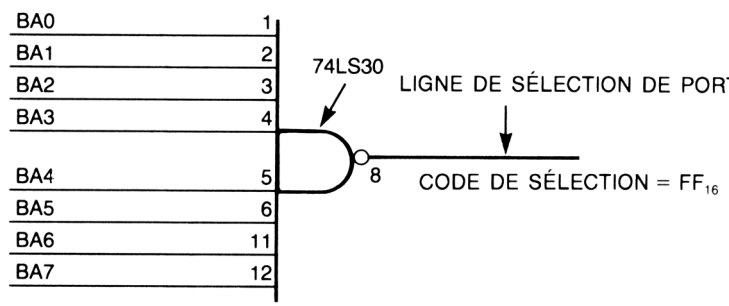


Figure 3.2 : *Schéma d'un circuit qui détecte FF comme adresse de port sur le bus d'adresses.*

On peut voir dans le schéma de la Figure 3.2 que la broche de sortie 8 du 74LS30 est au 0 logique si, et seulement si, toutes les lignes d'entrée sont au 1 logique. Remarquez que la broche de sortie 8 du 74LS30 (référencée *ligne de sélection de port*) est au 0 logique actif chaque fois que le bus d'adresses du système est logiquement égal au code de sélection distinct du port. Dans cet exemple, la ligne de sélection de port est active au 0 logique.

Il est important de se rappeler que la ligne de sélection de port peut devenir active même en l'absence de communications d'entrée-sortie. C'est parce que les mémoires du système utilisent les mêmes lignes d'adresses. Supposons par exemple que le microprocesseur soit en train de lire des données en adresse mémoire XXFFh et que la ligne de sélection de port devienne soudain active. Ceci peut se produire car les lignes d'adresses A0 à A7 donnent le code de sélection de port. En d'autres termes, si la ligne de sélection de port de votre système commence à signaler que le port est sélectionné à un moment où les instructions logicielles ne prévoient pas une communication du système avec le port, ne vous inquiétez pas, cette condition est valide.

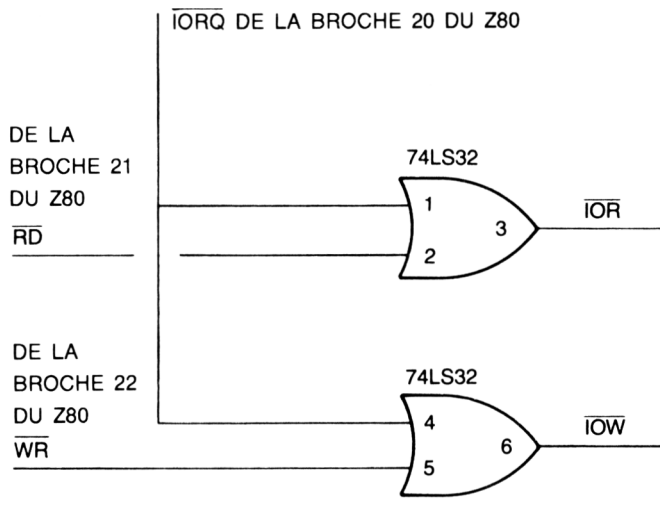


Figure 3.3 : *Schéma de circuit nécessaire pour produire les signaux de commandes IOR et IOW.*

3.3 - Production des commandes $\overline{\text{IOW}}$ et $\overline{\text{IOR}}$

Chaque fois que le Z80 effectue des opérations d'entrée-sortie, la ligne de commande référencée $\overline{\text{IORQ}}$ est activée. (Remarquez que cette commande est similaire au $\overline{\text{MREQ}}$ mentionné précédemment.) Les deux lignes de synchronisation $\overline{\text{RD}}$ et $\overline{\text{WR}}$ forment une combinaison logique avec $\overline{\text{IORQ}}$ pour donner les commandes $\overline{\text{IOR}}$ (*I/O Read*, lecture entrée-sortie) et $\overline{\text{IOW}}$ (*I/O Write*, écriture entrée-sortie).

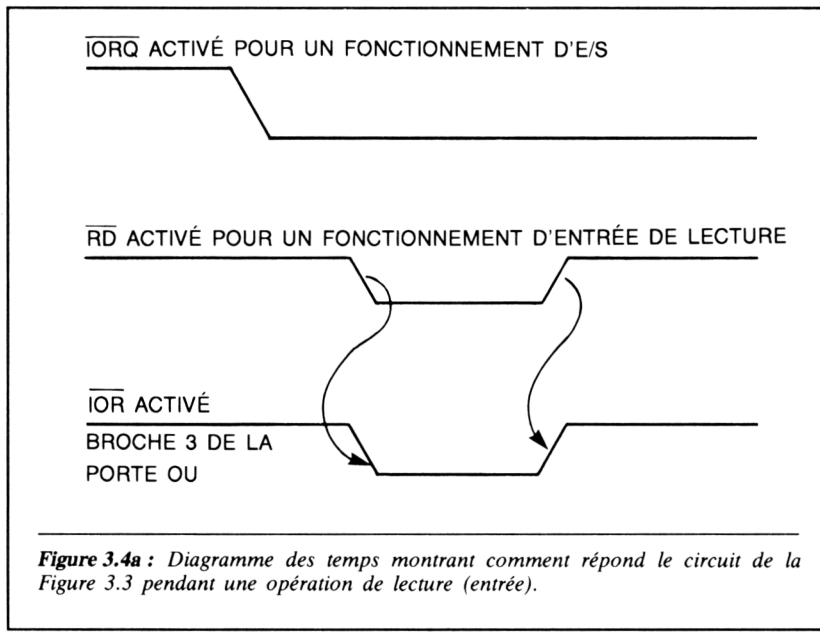


Figure 3.4a : *Diagramme des temps montrant comment répond le circuit de la Figure 3.3 pendant une opération de lecture (entrée).*

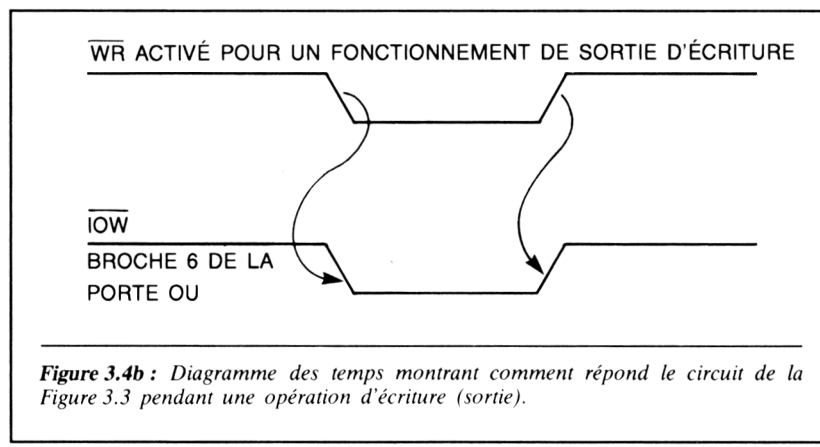


Figure 3.4b : *Diagramme des temps montrant comment répond le circuit de la Figure 3.3 pendant une opération d'écriture (sortie).*

La Figure 3.3 donne un schéma montrant comment sont produits, dans un système à Z80, les signaux de commande $\overline{IOR}$ et $\overline{IOW}$. Remarquez que ces signaux sont actifs pendant toutes les opérations d'entrée de lecture ou de sortie d'écriture, comme le montrent les Figures 3.4a et 3.4b.

Le signal d'écriture de port d'un système à Z80 est défini comme étant « le signal de validation d'écriture pour un port de sortie sélectionné ». Ce signal est produit chaque fois que la ligne de sélection de port et le signal de commande $\overline{IOW}$ sont actifs tous les deux. Le signal d'écriture de port

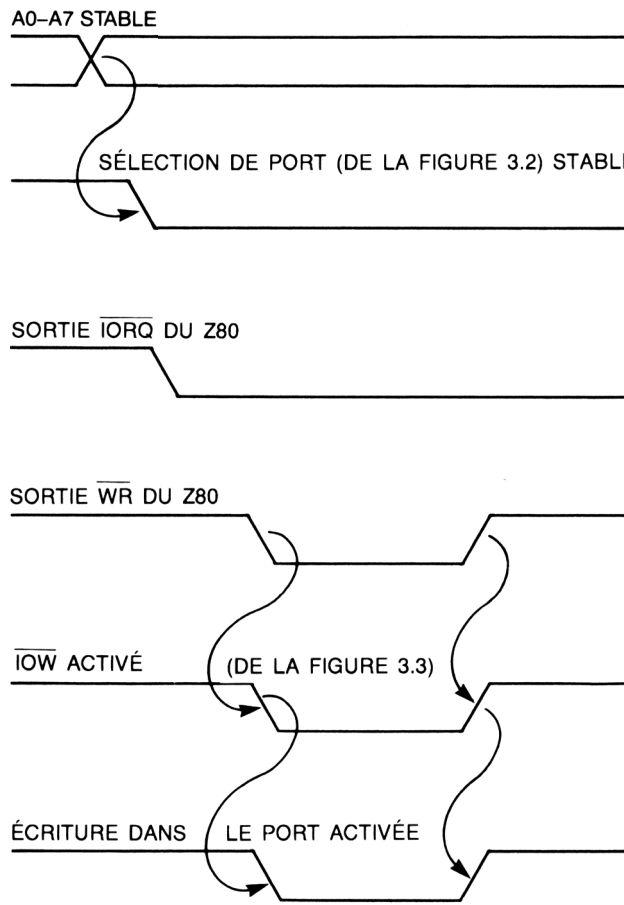


Figure 3.5 : *Déroulement des opérations pendant une sortie d'écriture.*

fournit le signal digital actif qui indique que les données provenant du microprocesseur doivent être verrouillées (par un *latch*) ou envoyées en écriture vers un port de sortie.

Le diagramme des temps de la Figure 3.5 montre le déroulement général des événements quand les données d'écriture sont envoyées vers un port de sortie. Dans cette figure, on peut considérer tous les signaux, excepté $\overline{WR}$, comme des niveaux logiques statiques. Le niveau du signal reste stable pendant toute l'opération. Nous avons employé ce concept et le diagramme des temps de la Figure 3.5 pour élaborer le schéma de la Figure 3.6. Il montre une façon de produire, par circuits, un signal de commande d'écriture de port.

Examinons ce diagramme et voyons comment fonctionne le circuit. La compréhension de ce fonctionnement vous aidera à mieux comprendre comment est utilisé chaque signal du Z80. (Rappelez-vous, cependant, qu'il y a d'autres façons de produire un signal de port.)

Le but du circuit de la Figure 3.6 est de fournir un signal actif au niveau 0 chaque fois que le CPU envoie des données d'écriture au port de sortie voulu. Dans cette figure, les lignes d'adresses du système A0 à A7 sont appliquées à un opérateur logique NAND à 8 entrées. C'est le même circuit que celui de la Figure 3.2.

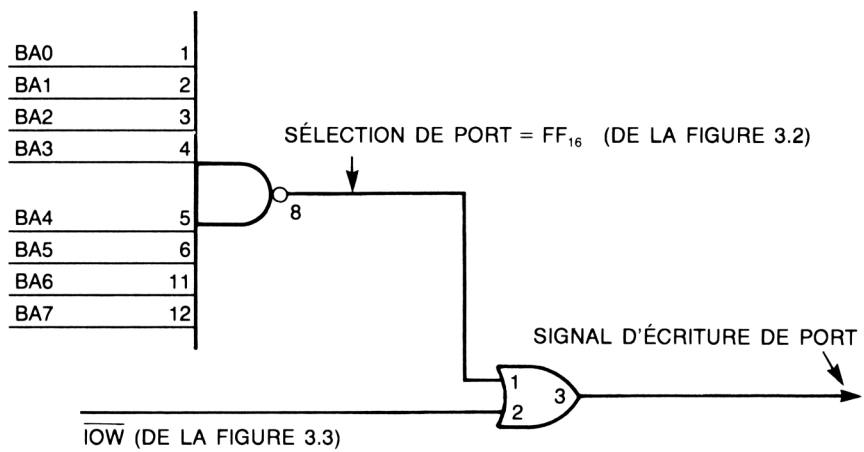


Figure 3.6 : *Schéma montrant les circuits permettant la réalisation du signal de commande d'écriture de port. Remarquez que ce signal est un signal de commande $\overline{IOW}$ validé. Il est validé par la ligne de sélection d'adresse de port.*

Quand toutes les lignes d'adresses, A0 à A7, sont au 1 logique (ce qui correspond au code OFF de sélection de port), la sortie de l'opérateur NAND donne un 0 logique. Cette ligne de sélection de port est connectée à une entrée de l'opérateur OU 74LS32. L'autre entrée de l'opérateur OU est connectée au signal $\overline{IOW}$ (montré Figure 3.3). Quand le signal $\overline{IOW}$ est au 0 logique, ceci indique que le CPU effectue la fonction consistant à envoyer un octet vers un port de sortie du système.

Si, par conséquent, le signal $\overline{IOW}$ est au 0 logique et que la sélection de port est au 0 logique, le CPU doit envoyer des données vers le port de sortie choisi. Nous avons, en effet, combiné le signal $\overline{IOW}$ au signal de sélection de port. Le résultat est un signal distinct, actif au 0 logique, qui a lieu si, et seulement si, le CPU du Z80 effectue une opération de sortie vers le port de sortie FF du système. Ce signal résultant s'appelle le *signal de commande d'écriture de port*. On l'emploie pour provoquer la prise en compte des données dans le port de sortie choisi.

Remarquez que ce n'est qu'une des façons d'effectuer une commande de port de sortie. Le déroulement des opérations et la manière d'employer le signal $\overline{IOW}$ sont cependant communs à la plupart des systèmes à microprocesseurs.

3.4 - Production du signal de lecture de port

Expliquons maintenant comment le Z80 lit les données dans un port d'entrée à 8 bits. Le diagramme des temps de la Figure 3.7 montre une séquence standard des différentes opérations qui se produisent au cours d'une lecture dans un port d'entrée. Dans ce diagramme, le signal $\overline{IOR}$ de sélection de port est semblable au signal $\overline{IOW}$ employé dans les opérations de sortie.

Examinons maintenant l'effet de signal $\overline{RD}$ de la Figure 3.7. $\overline{RD}$ est le signal de synchronisation qui produit le signal $\overline{IOR}$ pour le port d'entrée. Ce signal déclenche le transfert de données. Quand $\overline{RD}$ va au 0 logique, les données provenant du port d'entrée sont mises dans le bus de données du système et placées par le Z80 dans un registre interne. Un peu plus tard, $\overline{RD}$ va aller au 1 logique. A ce moment-là, les données du port d'entrée seront physiquement enlevées du bus de données du système et le transfert matériel sera terminé.

La Figure 3.8 montre l'une des façons de produire un signal de lecture de port, basée sur les conditions logiques données à la Figure 3.7. On peut voir dans la Figure 3.8 que ce circuit est très semblable à celui de la Figure 3.6. En fait, la seule différence entre ces deux circuits est que l'un utilise le signal

$\overline{\text{IOW}}$ et l'autre le signal $\overline{\text{IOR}}$. A part cela, ils sont identiques. Le signal de commande de lecture de port devient actif si, et seulement si, le Z80 reçoit physiquement des données du port spécifié, FF.

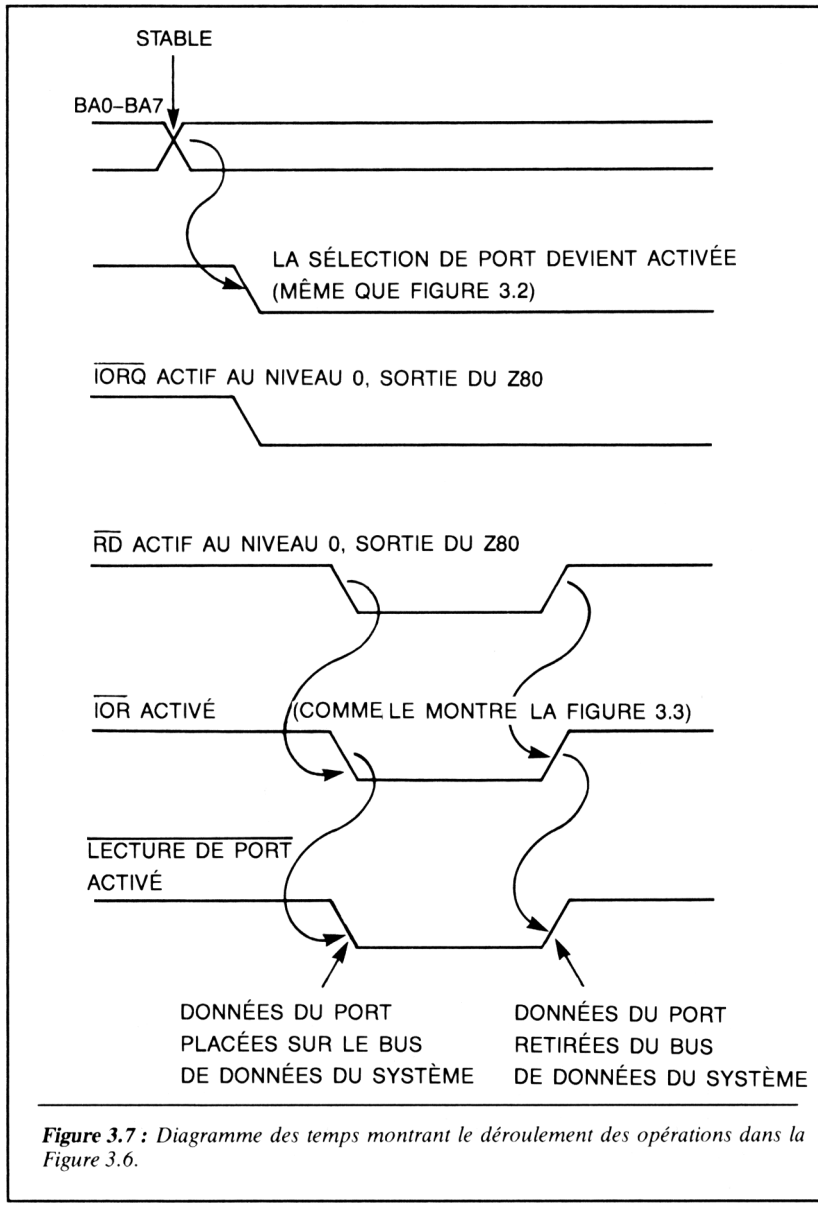


Figure 3.7 : *Diagramme des temps montrant le déroulement des opérations dans la Figure 3.6.*

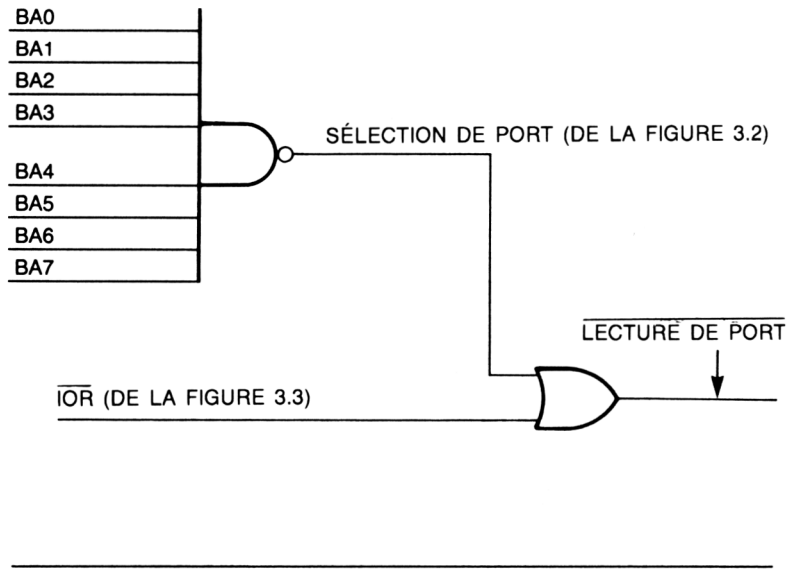


Figure 3.8 : *Schéma montrant les circuits nécessaires pour obtenir le signal de commande de lecture de port.*

3.5 - Schéma complet d'un port d'entrée-sortie

Nous allons maintenant nous occuper de ce qui se passe pendant les opérations de lecture et d'écriture d'entrée-sortie. Pour cette analyse, vous pouvez vous référer au schéma de la Figure 3.9, montrant un port d'entrée-sortie à 8 bits. Examinons d'abord comment le Z80 effectue une opération de sortie.

3.6 - Déroulement des opérations pendant une écriture en sortie

1. Pour une écriture en sortie, les lignes d'adresses A0 à A7 sont d'abord mises en configuration correspondant à l'adresse de sortie désirée, sous contrôle du Z80. A ce moment, les lignes d'adresses de sorties A0 à A7 sont décodées par le circuit de sélection de port. Dans la Figure 3.9, la broche de sortie 8 de IC1 devient active au niveau logique 0.

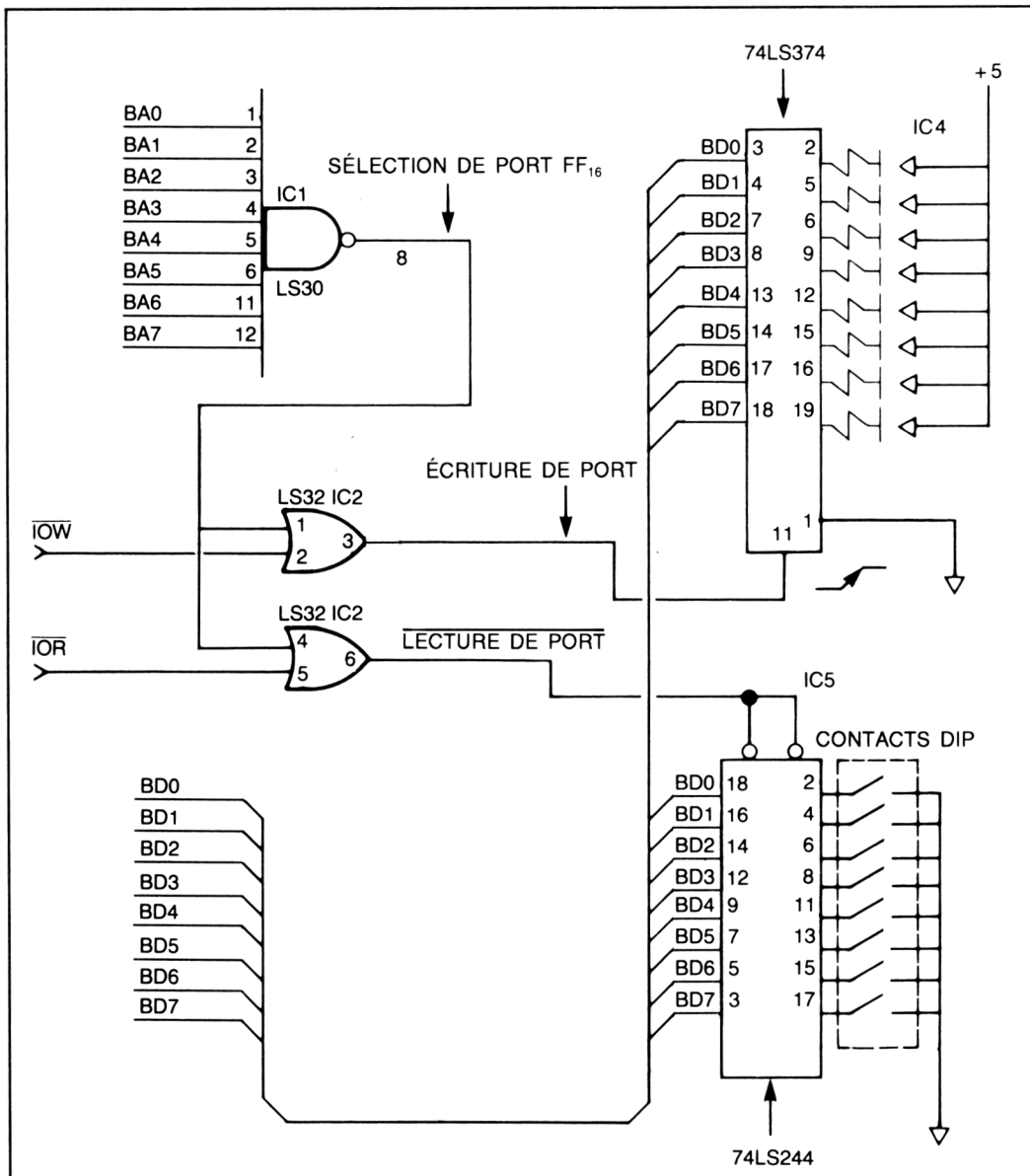


Figure 3.9 : *Schéma complet d'un port standard d'entrées et de sorties à 8 bits pour Z80. L'adresse de port pour le port d'entrées et de sorties est FF.*

2. Le Z80 envoie ensuite les données à écrire au port de sortie, par les lignes D0 à D7 du bus de données. Remarquez, dans la Figure 3.9, que les entrées de données D0 à D7 sont maintenant valides au *latch* octal 74LS374 (IC4).

3. La ligne de sortie $\overline{\text{IORQ}}$ du Z80 est ensuite mise au 0 logique. Ceci indique que le Z80 est en train d'effectuer une opération d'entrée-sortie. A ce moment-là, tout le décodage statique des signaux de sortie du Z80 est terminé.

4. La sortie de synchronisation $\overline{\text{WR}}$ du Z80 est ensuite mise au 0 logique. Cette action active la commande $\overline{\text{IOW}}$. Quand cette commande va au 0 logique, la commande d'écriture du port va aussi au 0 logique. La sortie $\overline{\text{WR}}$ du Z80 va ensuite au 1 logique. A ce moment-là, les données qui se trouvaient sur les lignes de données D0 à D7 sont écrites dans le 74LS374. Comme le signal est maintenant au 1 logique, l'opération de sortie est terminée.

Dans ce port de sortie, l'écriture des différentes données dans le port allume et éteint les LED, comme le montre la Figure 3.9. Un 0 logique écrit dans un bit de données spécifié allumera une LED ; un 1 logique écrit dans le port l'éteindra.

Les chapitres suivants montreront comment le fait d'écrire différentes données de sortie dans un port spécifié peut changer complètement le fonctionnement du port.

3.7 - Opération de lecture dans un port

Examinons maintenant le déroulement des opérations effectuées par le Z80 au cours d'une lecture de données dans un port d'entrée (Voir Figure 3.9). Les principales opérations rencontrées au cours d'une lecture d'entrée-sortie sont les mêmes que celles qui se produisent dans une écriture d'entrée-sortie, excepté que l'on emploie la commande de synchronisation $\overline{\text{RD}}$ plutôt que la commande $\overline{\text{WR}}$.

1. Les lignes d'adresses A0 à A7 sont d'abord mises sur le bus d'adresses du système. La logique de décodage, IC1, aura donc un 0 logique à la broche 8. Les données provenant du port d'entrée ne sont pas encore validées dans le bus de données du système. Ce circuit attend $\overline{\text{IORQ}}$ et le signal de commande synchronisée $\overline{\text{RD}}$ pour devenir actif.

2. $\overline{\text{IORQ}}$ est mis ensuite au 0 logique par le Z80. Cette action informe physiquement le système que le Z80 veut communiquer avec l'entrée-sortie plutôt qu'avec les mémoires.

3. Le signal de commande synchronisée $\overline{\text{RD}}$ est mis ensuite au 0 logique par le Z80. Ceci active la commande $\overline{\text{IOR}}$.

4. Une fois la commande $\overline{\text{IOR}}$ activée, le microprocesseur est prêt à recevoir des données du port d'entrée. Comme la sortie $\overline{\text{RD}}$ du Z80 est au 0 logique, les broches d'entrée de validation 1 et 19 du 74LS244, IC5, sont mises au 0 logique. Ceci permet aux sorties du 74LS244 de commander le bus de données du système. A partir de ce moment-là, toutes les valeurs logiques données par les contacts et appliquées aux tampons du 74LS244 sont maintenant entrées dans le bus de données du système. Pendant que les tampons du 74LS244 sont validés, le Z80 met les données du bus de données du système dans un registre interne.

5. Pour finir, $\overline{\text{RD}}$ est mis au 1 logique sous contrôle du Z80. Les données provenant du port d'entrée sont alors physiquement enlevées du bus de données du système. Le fonctionnement du système est maintenant terminé.

3.8 - Récapitulation des opérations

Révisons maintenant rapidement les opérations électriques qui se déroulent dans les circuits du système chaque fois que le Z80 effectue une opération d'entrée ou de sortie. Ceci nous servira plus tard quand nous examinerons en détail quelques dispositifs d'entrée-sortie.

3.9 - Séquence d'écriture en sortie

1. A0 à A7 sont établis aux valeurs de l'adresse correcte du port de sortie.

2. D0 à D7 sont établis aux valeurs des données à écrire dans le port de sortie.

3. $\overline{\text{IORQ}}$ est mis au 0 logique.

4. $\overline{\text{WR}}$ est mis au 0 logique.

5. $\overline{\text{WR}}$ est mis au 1 logique.

3.10 - Séquence de lecture en entrée

1. A0 à A7 sont établis aux valeurs de l'adresse correcte du port d'entrée.

2. $\overline{\text{IORQ}}$ est mis au 0 logique sous contrôle du Z80.

3. $\overline{\text{RD}}$ est mis au 0 logique.

4. $\overline{\text{RD}}$ est mis au 1 logique.

RESUME

Nous avons analysé, dans ce chapitre, les détails des opérations d'entrées et de sorties du Z80. Nous avons examiné plusieurs techniques de décodage et aussi ce qui se passe pendant une opération d'entrée et de sortie.

Si vous avez l'intention de dépanner ou d'établir des circuits pour un système à Z80, vous voudrez vous familiariser avec les opérations d'entrées et de sorties. Les renseignements donnés dans ce chapitre sont donc importants. Ils seront très utiles dans les chapitres suivants quand nous examinerons les circuits nécessaires pour interfacer le Z80 avec d'autres dispositifs et quand nous utiliserons des boîtiers spéciaux d'entrée-sortie LSI.

UTILISATION DE RAM DYNAMIQUES AVEC LE Z80

CHAPITRE 4

INTRODUCTION

Dans ce chapitre, nous allons montrer comment employer des RAM dynamiques avec un Z80. Nous allons d'abord examiner une RAM dynamique type. Nous montrerons ensuite comment les bus d'adresses, de données et de commandes du Z80 communiquent tous électriquement avec les RAM dynamiques. Nous examinerons, enfin, un système de RAM dynamique complet pour Z80.

La RAM dynamique que nous allons étudier dans ce chapitre est la 4116, de 16 K × 1. Nous avons choisi ce modèle parce qu'il a des caractéristiques générales de fonctionnement applicables à la plupart des RAM dynamiques utilisées actuellement. Une fois que vous aurez compris le fonctionnement de ce boîtier avec un Z80, vous comprendrez facilement comment fonctionnent la plupart des autres RAM dynamiques, y compris la 64 K × 1.

4.1 - Aperçu de la 4116

La 4116 est organisée en RAM de 16 384 (16 K) × 1. Elle utilise des lignes séparées d'entrée et de sortie de données. Avec ces caractéristiques, une mémoire a besoin de 14 lignes d'adresses (pour obtenir le total de 16 384 emplacements d'adresses), 1 ligne d'entrée de données et 1 ligne de sortie de données.

Pour pouvoir écrire des données dans cette RAM, il faut une ligne d'autorisation d'écriture. En plus, pour permettre à la 4116 de fonctionner, il lui faut des alimentations de + 12 v, + 5 v, − 5 v, et 0 v. Avec les différentes autres broches nécessaires, on a donc :

14 lignes d'adresses
1 ligne d'entrée de données
1 ligne de sortie de données
4 lignes d'alimentation
1 ligne d'autorisation d'écriture

soit au total 21 lignes, visibles sur la Figure 4.1.

La 4116 a un boîtier DIP (*Dual Inline Package*, boîtier à double rangée de broches), représenté en Figure 4.2. Bien que 16 broches pour ce boîtier puissent paraître insuffisantes, ce n'est pas une erreur. Les fabricants de ce modèle ont utilisé la technique de multiplexage pour économiser des broches. Le *multiplexage* consiste à entrer les 14 lignes d'adresses nécessaires en deux groupes séparés avec sept lignes dans chaque groupe. On entre un groupe de lignes d'adresses d'abord, puis on entre le second. Remarquez que la 4116 a sept broches de lignes d'adresses. Ces lignes sont référencées A0 à A6, comme le montre la Figure 4.3. Quand ces lignes introduisent les adresses d'ordre bas provenant du microprocesseur, soit A0 à A6, on les nomme *adresses de rangées* (ou *de lignes*). On les appelle *adresses de colonnes* quand les entrées d'adresses de la 4116 (A0 à A6) reçoivent les adresses d'ordre haut provenant du microprocesseur, soit A7 à A13 (Voir Figure 4.4).

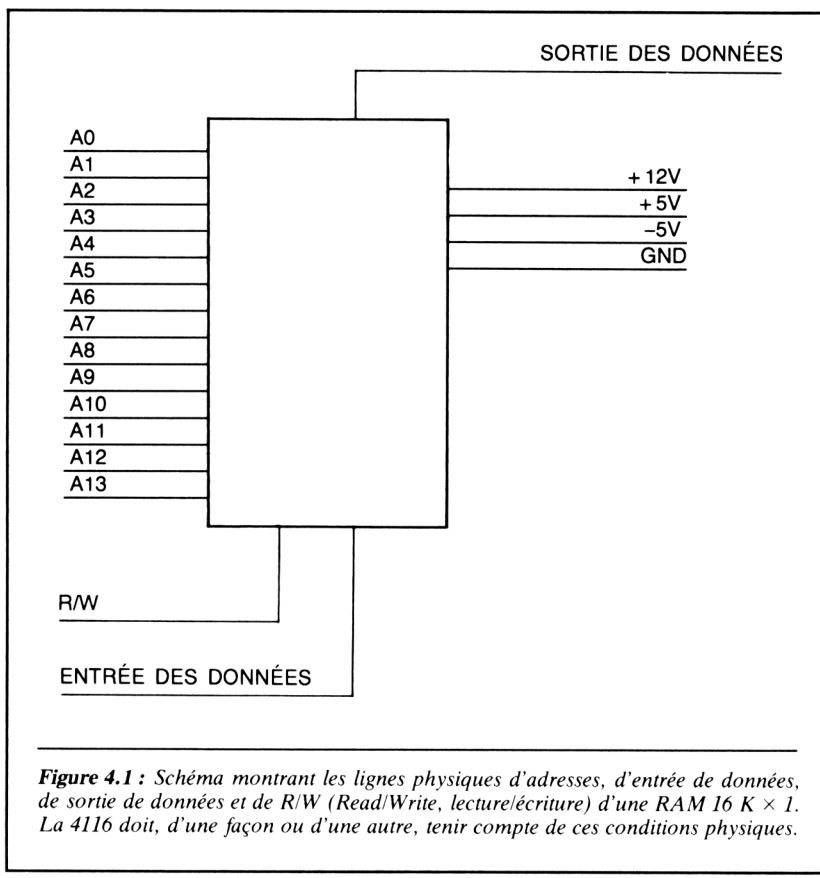


Figure 4.1 : *Schéma montrant les lignes physiques d'adresses, d'entrée de données, de sortie de données et de R/W (Read/Write, lecture/écriture) d'une RAM 16 K × 1. La 4116 doit, d'une façon ou d'une autre, tenir compte de ces conditions physiques.*

16,384 X 1-BIT DYNAMIC RAM

MK4116(J/N/E)-2/3

FEATURES

- Recognized industry standard 16-pin configuration from MOSTEK
- 150ns access time, 320ns cycle (MK 4116-2) 200ns access time, 375ns cycle (MK 4116-3)
- ± 10% tolerance on all power supplies (+12V, +5V)
- Low power: 462mW active, 20mW standby (max)
- Output data controlled by $\overline{CAS}$ and unlatched at end of cycle to allow two dimensional chip selection and extended page boundary
- Common I/O capability using "early write" operation
- Read-Modify-Write, $\overline{RAS}$-only refresh, and Page-mode capability
- All inputs TTL compatible,low capacitance, and protected against static charge
- 128 refresh cycles
- ECL compatible on VBB power supply (-5.7V)
- MKB version screened to MIL-STD-883
- JAN version available to MIL-M-38510/240

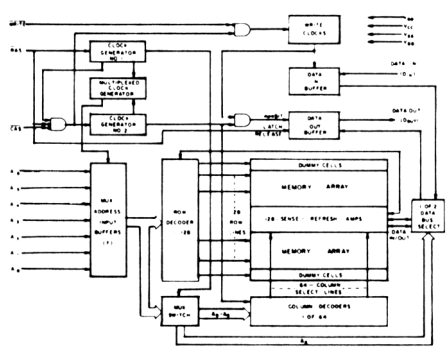
DESCRIPTION

The MK 4116 is a new generation MOS dynamic random access memory circuit organized as 16,384 words by 1 bit. As a state-of-the-art MOS memory device, the MK 4116 (16K RAM) incorporates advanced circuit techniques designed to provide wide operating margins, both internally and to the system user, while achieving performance levels in speed and power previously seen only in MOSTEK's high performance MK 4027 (4K RAM).

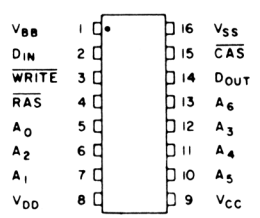
The technology used to fabricate the MK 4116 is MOSTEK's double-poly, N-channel silicon gate, POLY II™ process. This process, coupled with the use of a single transistor dynamic storage cell, provides the maximum possible circuit density and reliability, while maintaining high performance capability. The use of dynamic circuitry throughout, including sense amplifiers, assures that power dissipation is minimized without any sacrifice in speed or operating margin. These factors combine to make the MK 4116 a truly superior RAM product.

Multiplexed address inputs (a feature pioneered by MOSTEK for its 4K RAMS) permits the MK 4116 to be packaged in a standard 16-pin DIP. This recognized industry standard package configuration, while compatible with widely available automated testing and insertion equipment, provides highest possible system bit densities and simplifies system upgrade from 4K to 16K RAMs for new generation applications. Non-critical clock timing requirement allow use of the multiplexing technique while maintaining high performance.

FUNCTIONAL DIAGRAM



PIN CONNECTIONS



PIN NAMES

A_0-A_6	ADDRESS INPUTS	$\overline{WRITE}$	READ/WRITE INPUT
$\overline{CAS}$	COLUMN ADDRESS STROBE	V_{BB}	POWER (-5V)
D_{IN}	DATA IN	V_{CC}	POWER (+5V)
D_{OUT}	DATA OUT	V_{DD}	POWER (+12V)
$\overline{RAS}$	ROW ADDRESS STROBE	V_{SS}	GROUND

Figure 4.2 : Extrait de feuille de données indiquant les caractéristiques physiques et le brochage de la RAM dynamique 4116 de 16 K × 1.

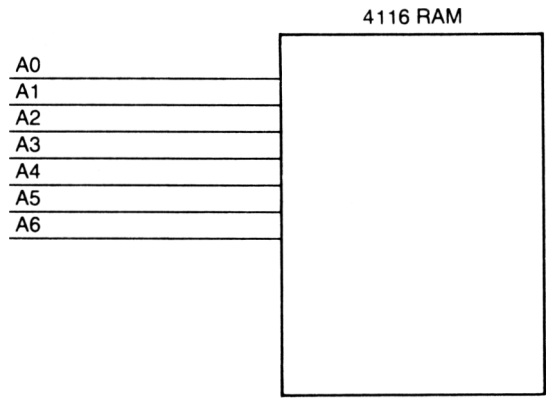


Figure 4.3 : *Schéma montrant le nombre réel de lignes d'entrées d'adresses de la 4116. Les 14 lignes d'entrées d'adresses sont toutes mises sur les 7 lignes effectives en deux groupes séparés.*

Les appellations adresses de rangées et adresses de colonnes proviennent de ce que ces groupes de lignes d'adresses effectuent des adressages correspondants dans la matrice interne de la mémoire. Les cellules de stockage d'une mémoire étant disposées dans une matrice, chacune de celles-ci a une adresse de rangée et de colonne qui sont distinctes. Grâce à un multiplexage, la 4116 n'a besoin que de 7 broches d'entrées d'adresses au lieu de 14.

Pour effectuer le multiplexage des signaux digitaux, il faut un signal qui indique le moment où les lignes d'entrées d'adresses correspondent à l'un de ces deux groupes. En d'autres termes, la 4116 doit être physiquement informée du moment où les informations appliquées aux broches d'entrées d'adresses A0 à A6 représentent l'adresse de rangée et du moment où elles représentent l'adresse de colonne. Dans ce but, il y a deux autres entrées. Elles sont référencées, dans la Figure 4.5, $\overline{\text{RAS}}$ (*Row Address Strobe*, commande des adresses de rangées) et $\overline{\text{CAS}}$ (*Column Address Strobe*, commande des adresses de colonnes). Quand l'entrée $\overline{\text{RAS}}$ est activée (0 logique), les informations présentes sur les lignes d'entrées d'adresses A0 à A6 sont prises en compte ou stockées intérieurement et utilisées comme les 7 bits d'ordre bas de l'adresse totale à 14 bits. Lorsque c'est l'entrée $\overline{\text{CAS}}$ du 4116 qui est activée (0 logique), ce sont les informations présentes sur les broches d'adresses mémoire A0 à A6 qui sont utilisées comme 7 bits d'ordre

haut de l'adresse totale à 14 bits. La Figure 4.5 montre un diagramme du temps avec $\overline{CAS}$ et $\overline{RAS}$ utilisés pour provoquer la prise en compte des adresses dans un boîtier 4116.

Les lignes d'adresses mémoire A0 à A6 de la 4116 sont des entrées. Cela signifie que le boîtier mémoire est, par lui-même, passif en ce qui concerne cette fonction. C'est à des circuits extérieurs de fournir les entrées d'adresses mémoire. En d'autres termes, des circuits extérieurs doivent délivrer à la 4116 ce qui suit : les entrées d'adresses de rangées, le $\overline{RAS}$, les entrées d'adresses de colonnes et le $\overline{CAS}$. De plus, les circuits extérieurs doivent synchroniser correctement ces signaux, avant leur utilisation par la mémoire.

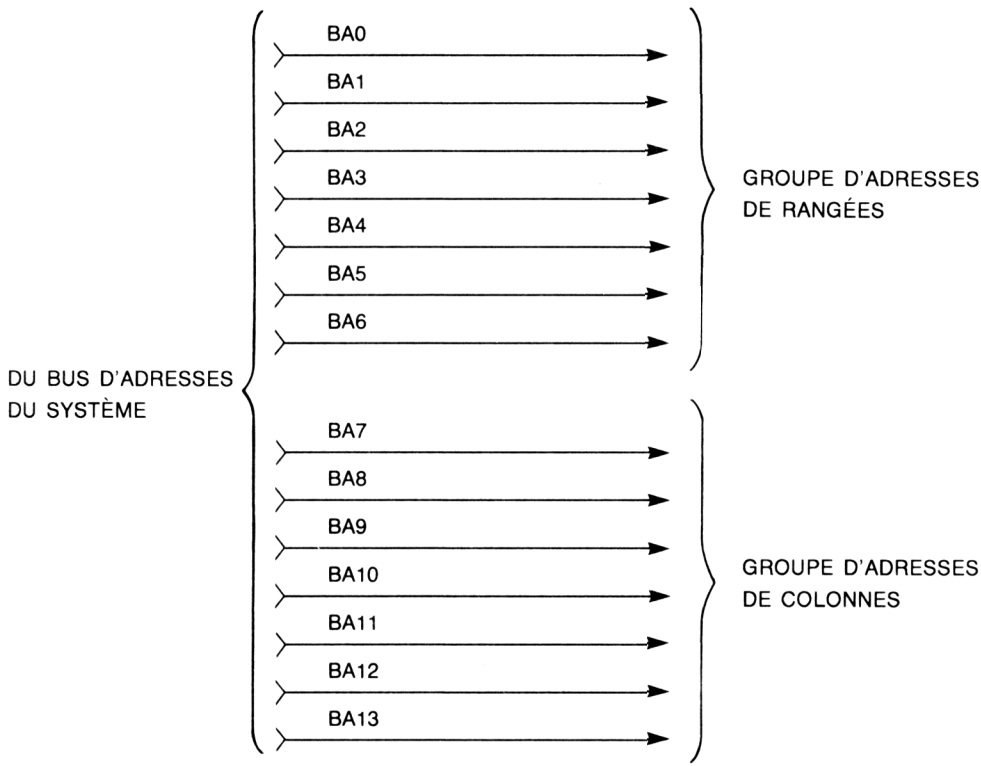


Figure 4.4 : *Les lignes d'adresses du sytème sont réparties en groupes d'adresses de rangées et en groupes d'adresses de colonnes. Les lignes d'adresses partent de la sortie des amplificateurs d'adresses du système.*

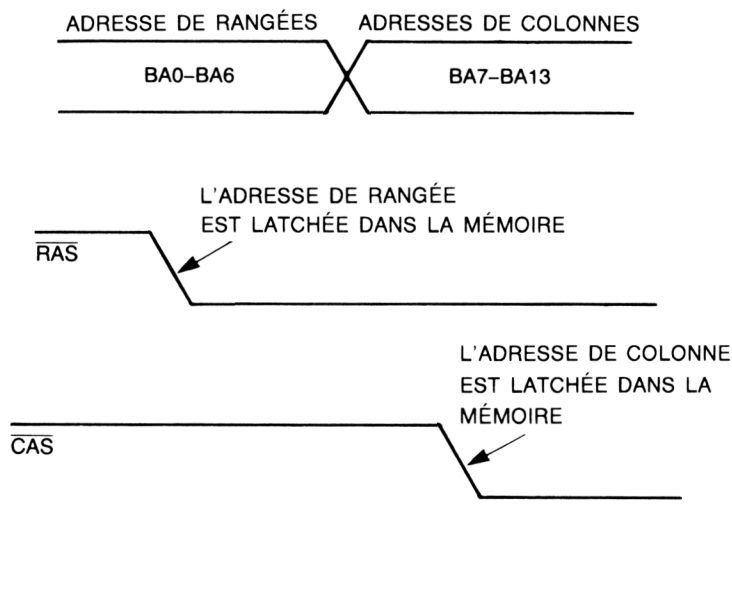


Figure 4.5 : *Diagramme des temps montrant les instants relatifs entre applications à la RAM 4116 du groupe d'adresses de rangées et du groupe d'adresses de colonnes. Remarquez que le signal $\overline{RAS}$ est actif pour le groupe de rangées et le signal $\overline{CAS}$ pour le groupe de colonnes.*

4.2 - Multiplexage des lignes d'adresses

On peut utiliser un circuit semblable à celui de la Figure 4.6 pour effectuer le multiplexage des entrées d'adresses de la 4116. Voyons comment le circuit fonctionne. Les 14 lignes d'adresses, A0 à A13, sont appliquées à des dispositifs digitaux appelés multiplexeurs. Quand la ligne de commande appliquée aux multiplexeurs est à l'état logique 1, les lignes d'adresses A0 à A6 du microprocesseur sont dirigées vers les broches d'entrées d'adresses A0 à A6 de la 4116. Quand la ligne de commande appliquée aux multiplexeurs est à l'état logique 0, ce sont les autres lignes d'adresses, soit A7 à A13, qui sont dirigées vers les entrées d'adresses A0 à A6 de la 4116.

Nous pouvons déduire de ces applications qu'il y a trois opérations électriques principales qui doivent être effectuées par des circuits pour introduire la totalité de l'adresse à 14 bits de la 4116. Ce sont :

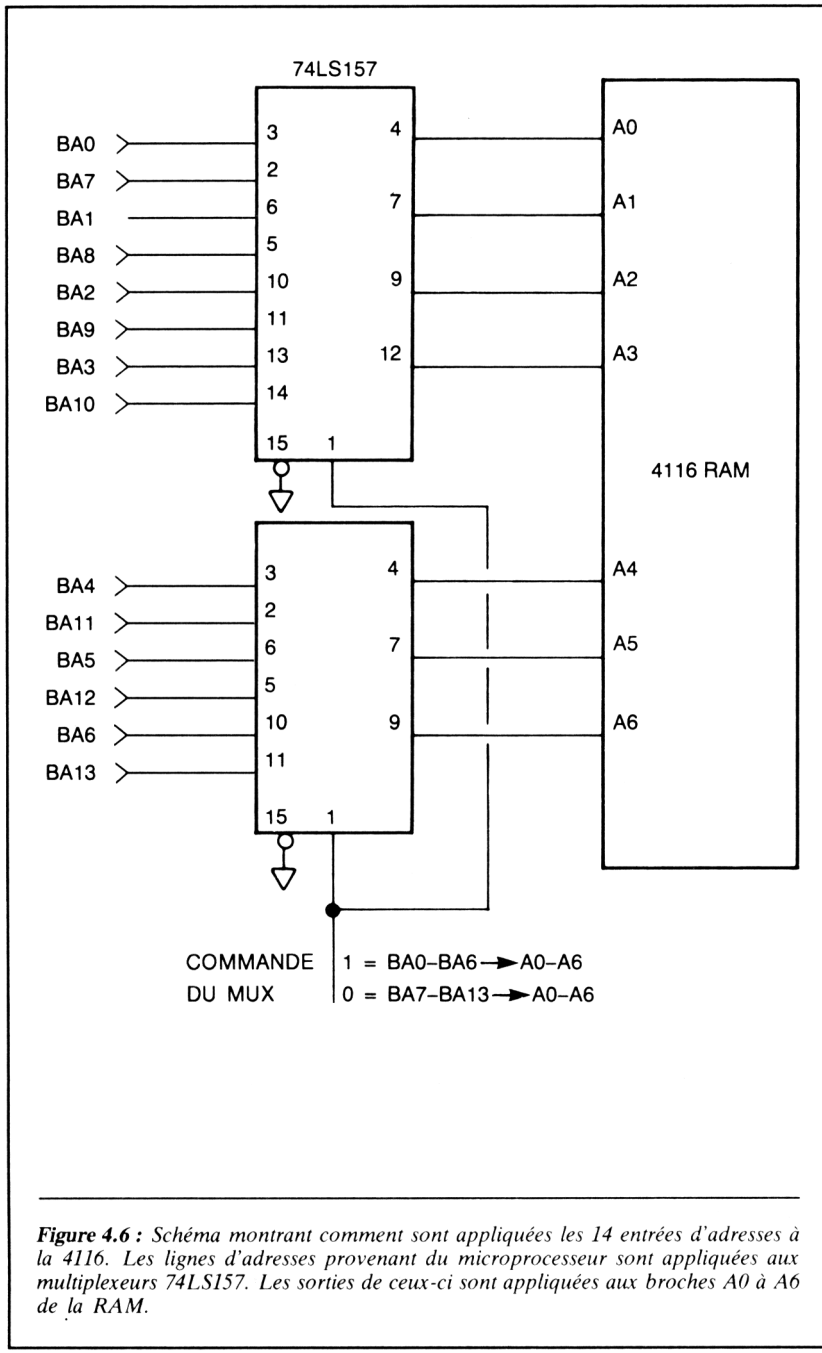


Figure 4.6 : *Schéma montrant comment sont appliquées les 14 entrées d'adresses à la 4116. Les lignes d'adresses provenant du microprocesseur sont appliquées aux multiplexeurs 74LS157. Les sorties de ceux-ci sont appliquées aux broches A0 à A6 de la RAM.*

1. Production du $\overline{RAS}$ au moment approprié.

2. Passage de la ligne de commande MUX au 0 logique, après le $\overline{RAS}$.

3. Production du $\overline{CAS}$ au moment approprié, après MUX.

La Figure 4.7 montre ces trois opérations dans un diagramme général des temps. (Nous verrons plus loin comment le Z80 les effectue.) Voyons maintenant comment on a introduit une adresse dans la 4116.

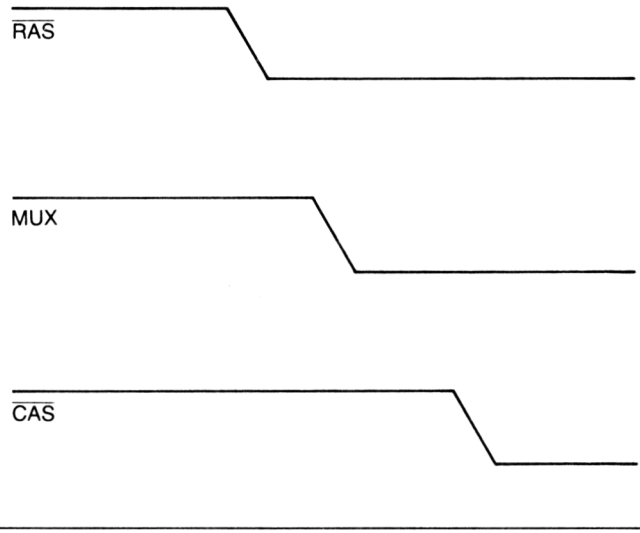


Figure 4.7 : *Diagramme des temps montrant les relations entre les signaux de commandes $\overline{RAS}$, MUX et $\overline{CAS}$ pour la RAM dynamique.*

4.3 - Schéma du système à RAM dynamiques 16 K × 8 bits

La Figure 4.8 montre le schéma d'un système à RAM dynamiques de 16 K × 8 bits pouvant être employé pour le Z80. Examinons-le maintenant et analysons les circuits.

En regardant ce schéma, il est important de se rappeler qu'il y a de nombreuses façons d'interfacer un microprocesseur avec des RAM dynamiques. Nous avons choisi la technique représentée ici parce qu'elle est simple et qu'elle peut être comprise par tous ceux qui n'ont qu'une expérience

limitée en circuits digitaux. Une interface avec une RAM dynamique peut être très compliquée notamment en cas d'utilisations industrielles. Cependant les conceptions les plus complexes découlent de ce qui est expliqué dans ce chapitre. Examinons la Figure 4.8.

L'adresse mémoire montrée dans la figure est introduite au moyen d'un circuit semblable à celui de la Figure 4.6 (ce bloc n'apparaît pas dans le schéma de la Figure 4.8). La ligne $\overline{RAS}$ est appliquée, en parallèle, à toutes les mémoires du système. L'entrée $\overline{CAS}$ appliquée au système de mémoires peut être activée ou non pendant le fonctionnement des mémoires (nous reprendrons ce point plus loin dans ce chapitre).

La logique permettant de rendre le signal $\overline{CAS}$ actif au moment approprié se trouve dans le bloc A. Cette ligne décode les états des lignes de sélection de mémoire et, au moment adéquat, active la ligne d'entrée $\overline{CAS}$ de la mémoire. La ligne de sélection de mémoire est produite par décodage de l'adresse du système.

L'espace d'adressage des RAM dynamiques est égal à 16 K octets. La ligne d'autorisation pour cet espace est décodée à partir des lignes

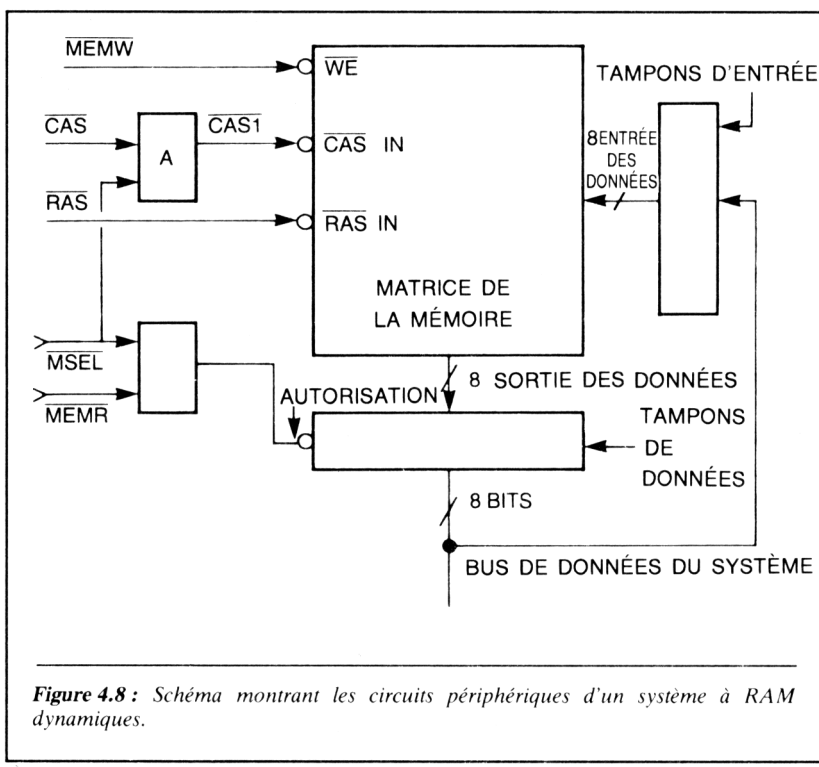


Figure 4.8 : *Schéma montrant les circuits périphériques d'un système à RAM dynamiques.*

d'adresses A14 et A15. Chaque fois que A14 et A15 sont au 1 logique, les RAM communiquent avec le CPU du Z80.

Dans le schéma de la Figure 4.8, la mémoire a son entrée d'autorisation d'écriture au 0 logique actif chaque fois que $\overline{MREQ}$ et $\overline{WR}$ sont tous les deux au 0 logique. On pourrait penser que cela présente des problèmes. Il n'en est rien. $\overline{MREQ}$ et $\overline{WR}$ sont tous deux au 0 logique chaque fois que le microprocesseur effectue une opération d'écriture dans n'importe quel espace d'adressage mémoire du système. Nous montrerons plus loin comment le système évite des opérations indésirables d'écriture en RAM dynamiques.

4.4 - Production des signaux $\overline{RAS}$, $\overline{CAS}$ et MUX

Nous allons maintenant voir comment le Z80 produit les lignes de commande $\overline{RAS}$, $\overline{CAS}$ et MUX. Beaucoup de techniques différentes peuvent être utilisées pour cela. Celle qui est décrite ici permet au débutant de comprendre exactement ce qui doit se passer et dans quel ordre. Et, en plus, elle fonctionne.

En nous reportant à la Figure 4.5, on peut voir l'ordre correct des signaux $\overline{RAS}$, MUX et $\overline{CAS}$. La Figure 4.9 montre comment on peut produire ces signaux dans le bon ordre. Le signal $\overline{RAS}$ est un signal $\overline{MREQ}$ amplifié. Souvenons-nous que le signal $\overline{RAS}$ doit provoquer la prise en compte des lignes d'adresses d'ordre bas, A0 à A6, du microprocesseur par le *latch* interne de la 4116. Remarquez que la ligne $\overline{MREQ}$ est active à l'instant approprié, c'est-à-dire après que l'adresse donnée par le microprocesseur soit stable sur le bus d'adresses.

Le but du signal MUX est d'aiguiller les lignes d'entrées d'adresses provenant des sorties A0 à A6 du microprocesseur vers les sorties A7 à A13. Ce signal est élaboré à partir de la ligne $\overline{RAS}$. En fait, le signal MUX est le signal $\overline{RAS}$ après qu'il soit passé par une ligne à retard. On peut employer n'importe quelle ligne à retard standard. Elle est parfois constituée de circuits digitaux mis en série avec la ligne $\overline{RAS}$. Les temps de propagation dans le (ou les) circuit(s) digital(aux), permettent d'employer celui-ci comme une ligne à retard. Le temps de retard effectif entre $\overline{RAS}$ et MUX est d'environ 50 ns.

Après que la ligne MUX soit passée au 0 logique actif, la ligne $\overline{CAS}$ peut être activée. On emploie également une ligne à retard pour produire le signal $\overline{CAS}$.

Cette technique de production de signaux est facile à réaliser avec des circuits. Il y a cependant quelques difficultés. La principale est due au fait que la mémoire doit avoir un temps d'accès très court pour pouvoir fonctionner à la vitesse maximale du microprocesseur, car son temps

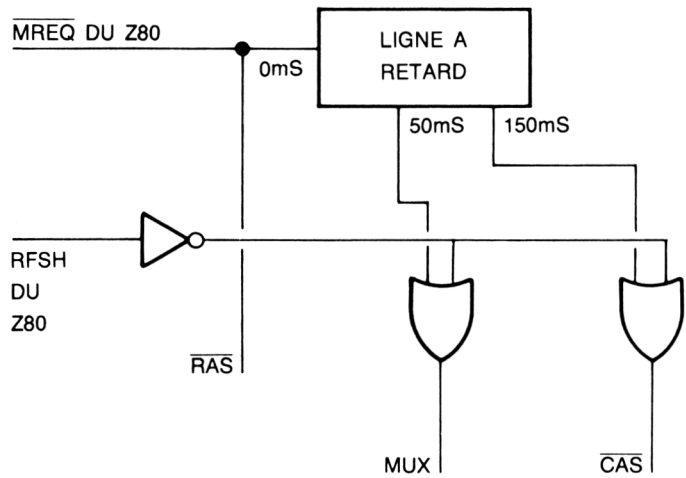


Figure 4.9 : *Schéma montrant comment sont élaborés les signaux de commandes* $\overline{RAS}$, $\overline{CAS}$ *et MUX.*

d'accès se mesure à partir du front descendant de $\overline{CAS}$. Une manière de résoudre ce problème est de ralentir l'horloge du système.

Bien que la fréquence d'horloge du système doive être suffisamment basse pour tenir compte du temps d'accès de la mémoire, elle ne doit pas être trop basse, pour des considérations de rafraîchissement (qui seront expliquées plus loin dans ce chapitre). La Figure 4.9 montre un diagramme des temps exacts pour le Z80. Ce diagramme donne les temps spécifiques de chaque signal. Il montre aussi les séquences et comment certains signaux du Z80 servent à produire les entrées de RAM, $\overline{RAS}$ et $\overline{CAS}$.

4.5 - Entrée des données dans la RAM dynamique

La RAM dynamique 4116 utilise des entrées-sorties séparées. Souvenons-nous que des broches distinctes sont affectées à l'entrée et à la sortie des données. Voyons maintenant comment les données en provenance du microprocesseur sont introduites dans la RAM.

La Figure 4.10 présente un amplificateur qui relie le bus de données du système aux broches d'entrée de données des 4116. Cette figure montre que les amplificateurs sont toujours validés. Les données provenant du bus de données du système apparaissent (électriquement), à tout instant, sur les broches d'entrée des RAM. Ceci est physiquement possible car les données ne seront pas prises en compte par les RAM tant qu'un signal de validation d'écriture ne sera pas activé.

La question qui se pose tout naturellement est « Pourquoi donc utiliser des amplificateurs ? » Compte tenu de ce que les données du bus de données sont introduites telles quelles dans les entrées de RAM, il semblerait que l'on puisse se passer de l'amplificateur. Il est exact que certains systèmes ne comportent pas d'amplificateurs de données ; il y a pourtant deux raisons qui militent en faveur de l'emploi d'amplificateurs :

- pour isoler les entrées des RAM du bus de données dans son ensemble. Ceci aide lors d'une recherche de panne dans un système défectueux. Si un boîtier RAM distinct est défectueux, tout le bus de données peut en être affecté. Ceci est même pire quand plusieurs boîtiers RAM sont reliés au bus de données du système.

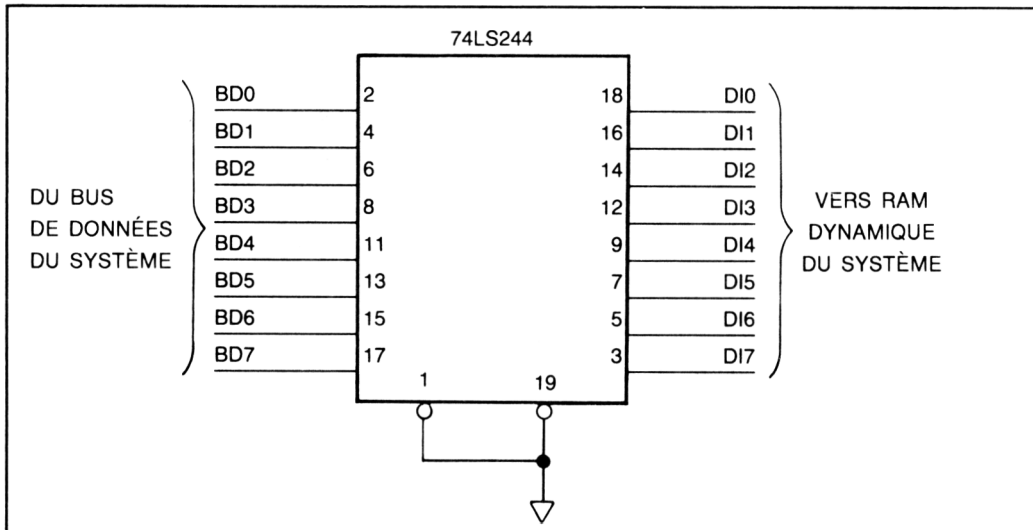


Figure 4.10 : *Circuit nécessaire pour entrer les données dans les lignes d'entrées des RAM. Cependant, dans certains systèmes, on peut ne pas utiliser cet amplificateur d'entrées. Les entrées des RAM dynamiques sont alors connectées directement au bus de données du système.*

- pour contrôler le bruit. Le bus de données du système est généralement très « bruité », autrement dit il y circule des signaux parasites indésirables. Les amplificateurs du bus de données aident à empêcher le bruit du bus de données du système d'atteindre les broches d'entrées des RAM.

La plupart des fabricants recommandent l'emploi d'amplificateurs d'entrées de données.

4.6 - Ecriture de données dans une RAM dynamique

Nous allons maintenant apprendre comment le microprocesseur écrit des données dans la 4116. Le schéma de la Figure 4.8 montre que les signaux suivants doivent être actifs pour permettre une opération d'écriture en mémoire :

- $\overline{\text{MREQ}}$ doit être au 0 logique
- $\overline{\text{WR}}$ doit être au 0 logique

Ces deux signaux activent la validation d'écriture des mémoires du système. En outre, on emploie les signaux suivants pour indiquer que l'espace mémoire est physiquement sélectionné :

- A14 et A15 doivent être au 1 logique.
- La RAM dynamique doit occuper l'espace d'adressage compris entre C000 et FFFF.

Dans ces conditions, $\overline{\text{CAS}}$ est activé et une opération d'écriture se produit.

Examinons le diagramme des temps pour une opération d'écriture en mémoire ; celle-ci peut avoir lieu dans deux cas différents : quand la mémoire est validée et quand elle ne l'est pas. Quand la mémoire n'est pas validée, le microprocesseur écrit des données dans un autre espace mémoire ou entrée-sortie.

La Figure 4.11 montre le déroulement principal d'une opération d'écriture en mémoire quand la RAM dynamique est validée. La première opération qui se produit est la validation de la ligne de sélection de mémoire. Celle-ci a lieu quand les lignes d'adresses A14 et A15 vont au 1 logique sous le contrôle du Z80. Juste après cela, $\overline{\text{RAS}}$ est activé vers le $\overline{\text{RAS}}$ du système. Ce signal est produit par la ligne de commande $\overline{\text{MREQ}}$ provenant du Z80. Après une activation du $\overline{\text{RAS}}$, la commande MUX d'adresses va au 0 logique après un retard de 50 nanosecondes à partir du front descendant de $\overline{\text{RAS}}$.

Quand la commande MUX va au 0 logique, les entrées d'adresses de la 4116 passent de l'adressage de rangées à l'adressage de colonnes. Après la commutation du signal MUX, $\overline{CAS}$ est activé. Le signal $\overline{CAS}$ est produit en retardant MUX. L'entrée $\overline{WE}$ de la mémoire est ensuite produite par la ligne $\overline{WR}$ provenant du Z80.

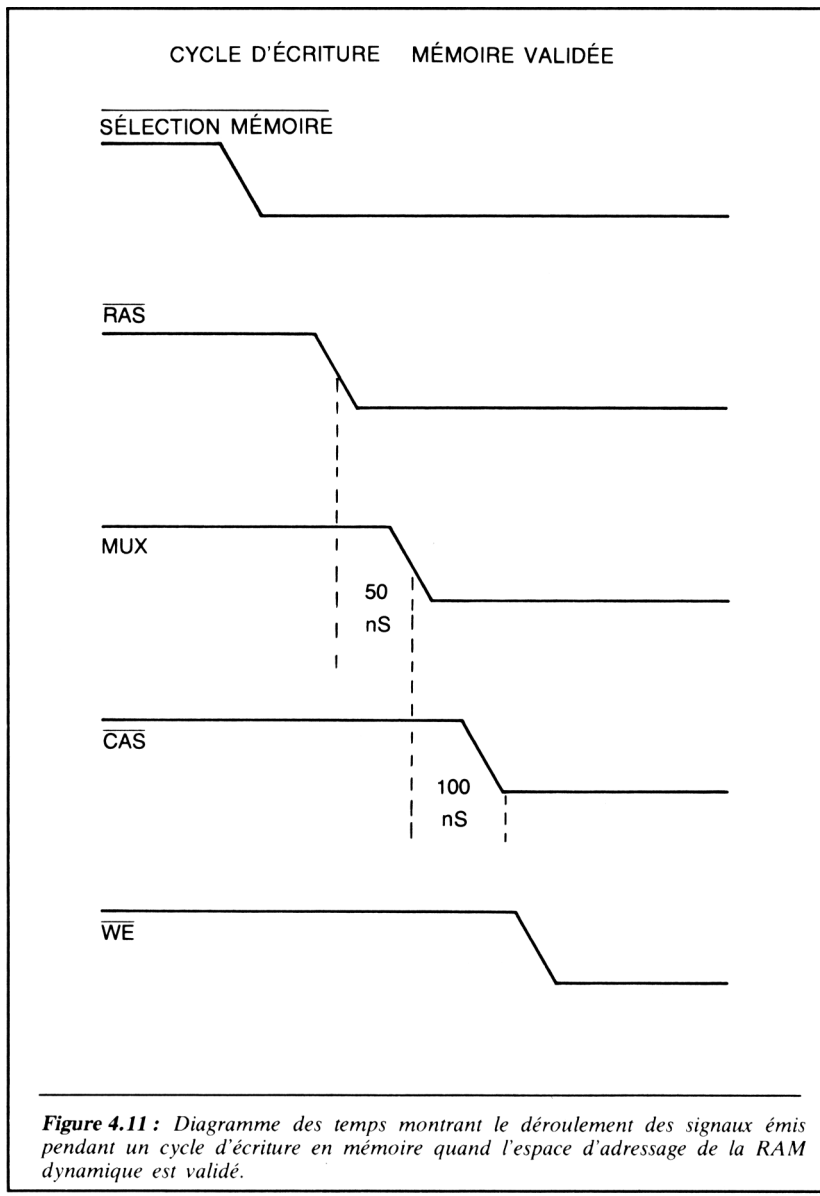


Figure 4.11 : *Diagramme des temps montrant le déroulement des signaux émis pendant un cycle d'écriture en mémoire quand l'espace d'adressage de la RAM dynamique est validé.*

Le diagramme des temps de la Figure 4.11 montre les opérations qui se déroulent chaque fois que le microprocesseur écrit des données dans la RAM dynamique. La Figure 4.12 montre un diagramme des temps relatif aux opérations qui se déroulent chaque fois que le microprocesseur écrit des données dans une mémoire qui ne fait pas partie de l'espace d'adressage des RAM dynamiques. En d'autres termes, comme les données sont destinées à être écrites ailleurs, la section des RAM dynamiques n'est pas validée.

Examinons le diagramme des temps de la Figure 4.12. Dans cette figure la ligne de sélection de mémoire n'est pas active. Ceci est dû au fait que A14 et A15, *tous les deux*, ne sont pas au 1 logique. $\overline{\text{RAS}}$ est actif parce que la ligne de commande $\overline{\text{MREQ}}$ du Z80 est active. La ligne de commande MUX va au 0 logique parce qu'elle dépend de la ligne de commande $\overline{\text{RAS}}$. La ligne $\overline{\text{CAS}}$ du système est activée, mais $\overline{\text{CAS1}}$ relié à la mémoire, n'est pas produit. Ceci est dû au fait que la ligne de sélection de mémoire n'est pas active.

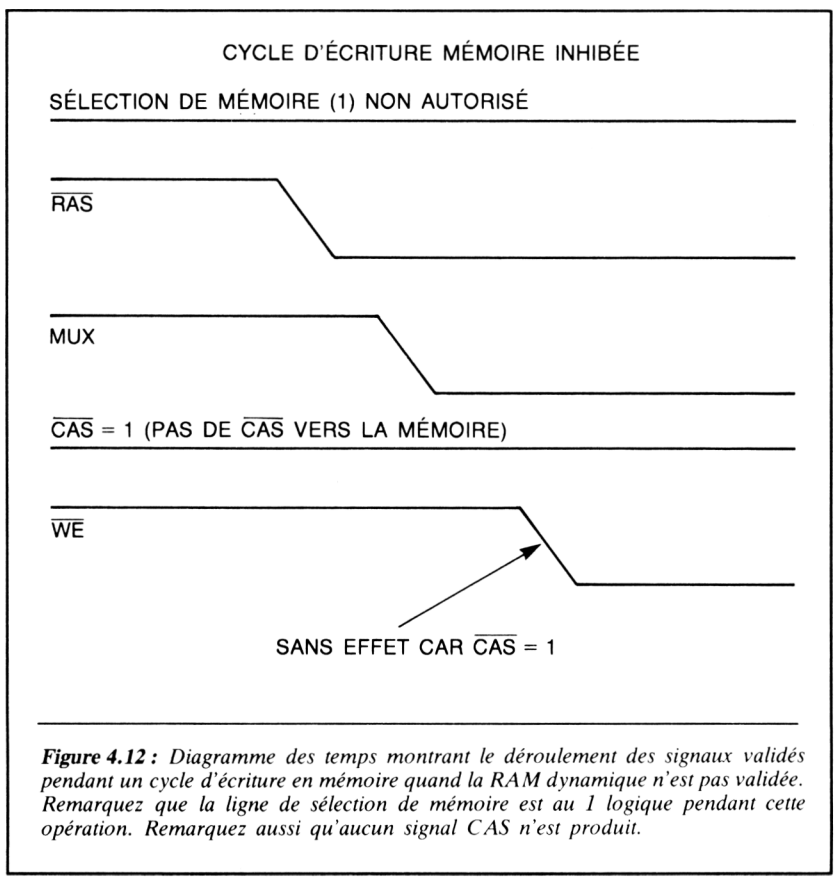


Figure 4.12 : *Diagramme des temps montrant le déroulement des signaux validés pendant un cycle d'écriture en mémoire quand la RAM dynamique n'est pas validée. Remarquez que la ligne de sélection de mémoire est au 1 logique pendant cette opération. Remarquez aussi qu'aucun signal CAS n'est produit.*

Pour finir, $\overline{WE}$ du système de RAM dynamiques devient actif. Cependant, les données ne sont pas transférées dans les RAM, car les entrées $\overline{CAS}$ et $\overline{RAS}$ des 4116 doivent être toutes les deux au 0 logique actif pour valider électriquement le boîtier mémoire, invalidant ainsi l'espace mémoire pour toute écriture en mémoire, quand son espace d'adressage n'est pas sélectionné par le microprocesseur.

4.7 - Sortie des données des RAM dynamiques

Nous allons maintenant apprendre comment la ligne de sortie de données de la RAM dynamique est mise sur le bus de données du système pendant une opération de lecture en mémoire. Cette technique est semblable à celle permettant de mettre les données des RAM statiques, ROM et dispositifs d'entrée-sortie, dans le bus de données du système. La Figure 4.13 montre un circuit qui peut effectuer cela.

Dans cette figure, on utilise un amplificateur unidirectionnel 74LS244. Les entrées des amplificateurs sont reliées aux broches de sorties de données des RAM 4116. Les sorties de la 74LS244 sont connectées directement au bus de données du système. Quand le microprocesseur réclame des données aux RAM, les amplicateurs sont autorisés et mettent les données des RAM dynamiques dans le bus de données du système.

L'adressage de la RAM pour une opération de lecture en mémoire est identique à celui utilisé pour une opération d'écriture. Les signaux $\overline{RAS}$, MUX et $\overline{CAS}$ ont exactement le même *timing*. Pour autoriser les sorties de données des RAM dynamiques dans le bus de données du système pendant une opération de lecture en mémoire, les opérations physiques suivantes doivent avoir lieu :

1. La sortie $\overline{MREQ}$ du Z80 doit être au 0 logique, ce qui indique une opération de lecture en mémoire.

2. A15 et A14 doivent être tous les deux au 1 logique, ce qui active la sélection de mémoire pour l'espace d'adressage des RAM dynamiques.

3. La sortie $\overline{RD}$ du Z80 doit être au 0 logique, ce qui indique une opération de lecture sur le bus de données du système à Z80. Ce signal indique physiquement que le Z80 est prêt à recevoir des données.

Comme on peut le voir sur la Figure 4.13, quand $\overline{MREQ}$ est au 0 logique et A14 et A15 tous les deux au 1 logique, les entrées appliquées aux broches 1 et 2 de l'opérateur logique OU sont au 0 logique. Quand $\overline{RD}$ est activé au 0 logique, la 74LS244 est validée. Le microprocesseur maintiendra alors $\overline{RD}$

au 0 logique pendant un temps qui dépend de la fréquence d'horloge du Z80. Quand le Z80 met $\overline{RD}$ au 1 logique, les données des RAM sont physiquement ôtées du bus de données du système. Ceci termine le transfert de mémoire.

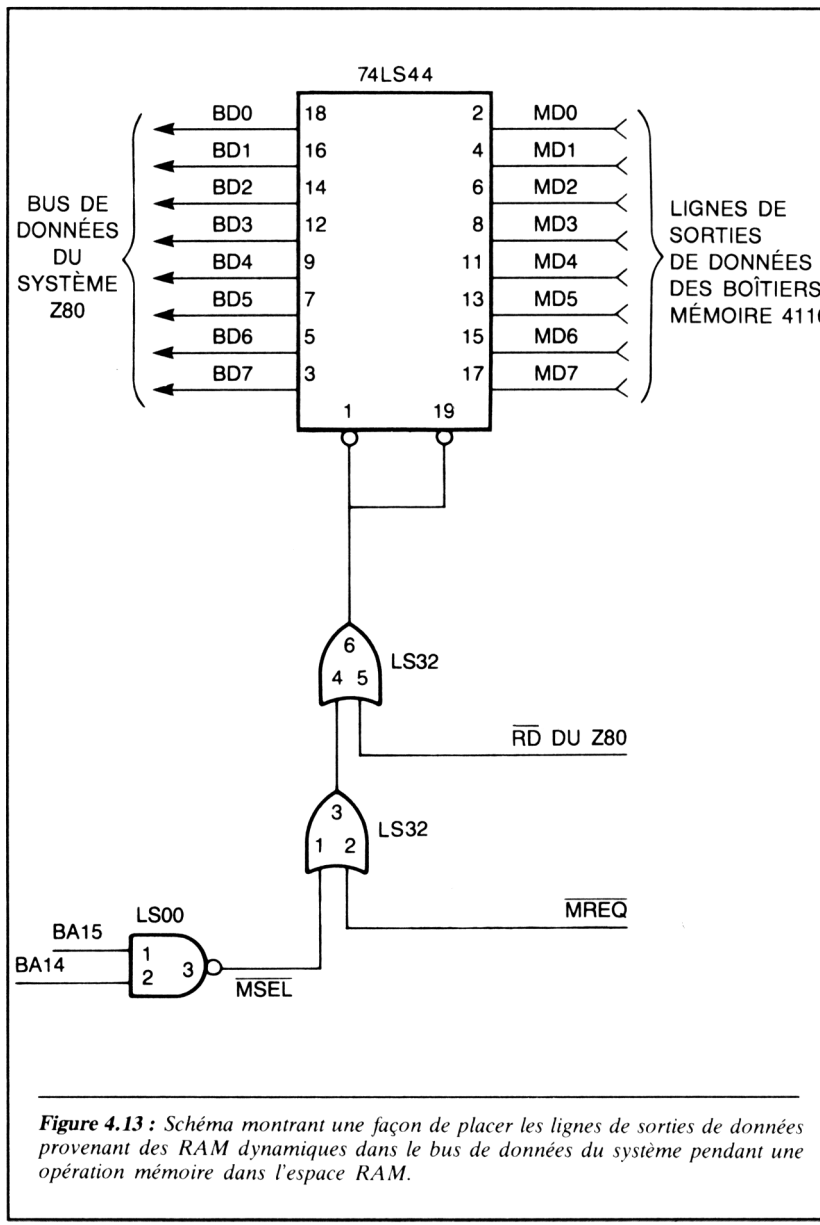


Figure 4.13 : *Schéma montrant une façon de placer les lignes de sorties de données provenant des RAM dynamiques dans le bus de données du système pendant une opération mémoire dans l'espace RAM.*

4.8 - Rafraîchissement des RAM dynamiques

En examinant jusqu'à présent les RAM dynamiques, nous avons omis d'indiquer qu'elles doivent être rafraîchies. *Rafraîchissement* signifie que les informations contenues dans les cellules de stockage internes de la RAM dynamique. La technique exposée ici est utilisée par le Z80. Disons que, garder leur validité. D'une façon générale, l'information contenue dans une cellule de stockage de RAM dynamique reste valide pendant seulement quelques (généralement deux) millisecondes ; si la cellule n'est pas lue pendant ce laps de temps, la donnée sera perdue.

Il y a plusieurs techniques pour rafraîchir les données dans une RAM dynamique. La technique exposée ici est utilisée par le Z80. Disons que, pour que l'information interne soit conservée, la RAM dynamique doit provoquer l'accès à chaque cellule de stockage toutes les deux millisecondes. Cette lecture doit se produire même quand l'espace de RAM dynamique n'est pas utilisé pour l'exécution du programme.

Lorsque des données sont lues dans la 4116, une rangée entière de cellules est rafraîchie en parallèle. Une RAM complète peut être rafraîchie en lisant toutes les entrées d'adresses A0 à A6. Ceci correspond aux entrées d'adresses des 128 rangées. On doit balayer chaque adresse de rangée une fois toutes les deux millisecondes. Donc, comme il y a 128 adresses de rangées distinctes à lire toutes les deux millisecondes, le microprocesseur doit lire une adresse de rangée différente toutes les 16 microsecondes.

Le microprocesseur Z80 est un microprocesseur très intéressant à utiliser avec des RAM dynamiques, parce qu'il a un dispositif incorporé nommé *compteur de rafraîchissement* qui permet des interfaces très simples avec les mémoires dynamiques. On peut établir ce compteur pour qu'il émette une adresse de rafraîchissement sur le bus d'adresses du système à n'importe quel intervalle de temps donné. Après avoir émis cette adresse, le Z80 incrémente automatiquement le compteur pour émettre l'adresse suivante à l'instant spécifié.

Ce système interne du Z80 s'occupe automatiquement du système de RAM dynamiques. Durant la phase de recherche de code opération d'un cycle d'instructions, l'adresse de rafraîchissement est sortie sur les lignes d'adresses A0 à A6 du Z80. Cette adresse est émise pendant le temps T4, utilisé intérieurement par le Z80 pour décoder le code opération qui vient d'être extrait d'une mémoire. Ce type d'adressage de rafraîchissement s'appelle *vol de cycle.*

Pendant le laps de temps où l'adresse de rafraîchissement est émise par le Z80, sa broche RFSH est active. Pour la RAM dynamique, le cycle de rafraîchissement demande que seul $\overline{\text{RAS}}$ soit activé. (Rappelons-nous que la ligne de commande $\overline{\text{RAS}}$ valide l'adresse de rangée dans le système de mémoire.) On ne veut pas activer la ligne de commande $\overline{\text{CAS}}$ parce que la

mémoire serait validée et que pour effectuer l'opération de rafraîchissement interne, la mémoire ne demande pas que l'entrée $\overline{CAS}$ soit active. (Vous pouvez vous référer à une feuille de caractéristiques complètes de 4116 pour voir le *timing* exact d'une opération de rafraîchissement.)

La Figure 4.14 expose le déroulement des opérations de rafraîchissement de la 4116 ; elle montre que l'adresse de rafraîchissement est émise sur les lignes d'adresses A0 à A6 du Z80 et que le compteur interne de rafraîchissement incrémente automatiquement de un l'adresse de rafraîchissement à chaque cycle de rafraîchissement.

Pendant un cycle de rafraîchissement, le Z80 active la ligne $\overline{MREQ}$. Ceci provoque $\overline{RAS}$. Néanmoins, pendant un cycle de rafraîchissement, la ligne de commande $\overline{RD}$ n'est pas activée. La logique du circuit de $\overline{CAS}$ l'empêche de l'être. De plus, l'absence de $\overline{RD}$ empêche les amplificateurs de sortie trois-états d'être validés et donc de mettre des données de la mémoire dans le bus de données du système.

Il faut remarquer que le rafraîchissement des mémoires a lieu tant que le Z80 est en train d'exécuter un programme, c'est-à-dire aussi longtemps qu'il recherche des codes opération.

Cependant l'intervalle de temps entre recherches de codes opération peut ne pas être supérieur à l'intervalle de temps nécessité par le rafraîchissement de la mémoire. Certaines instructions sont plus longues à exécuter que

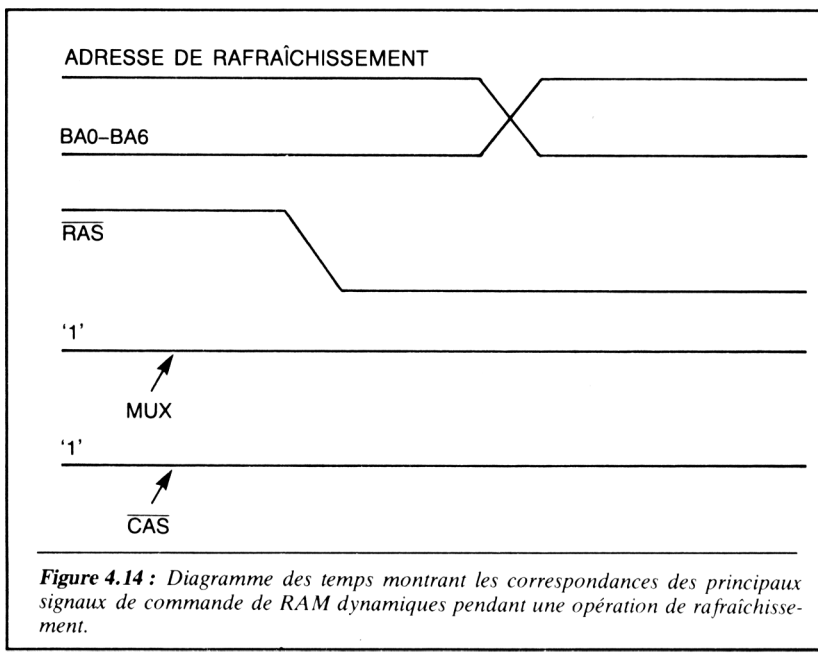


Figure 4.14 : *Diagramme des temps montrant les correspondances des principaux signaux de commande de RAM dynamiques pendant une opération de rafraîchissement.*

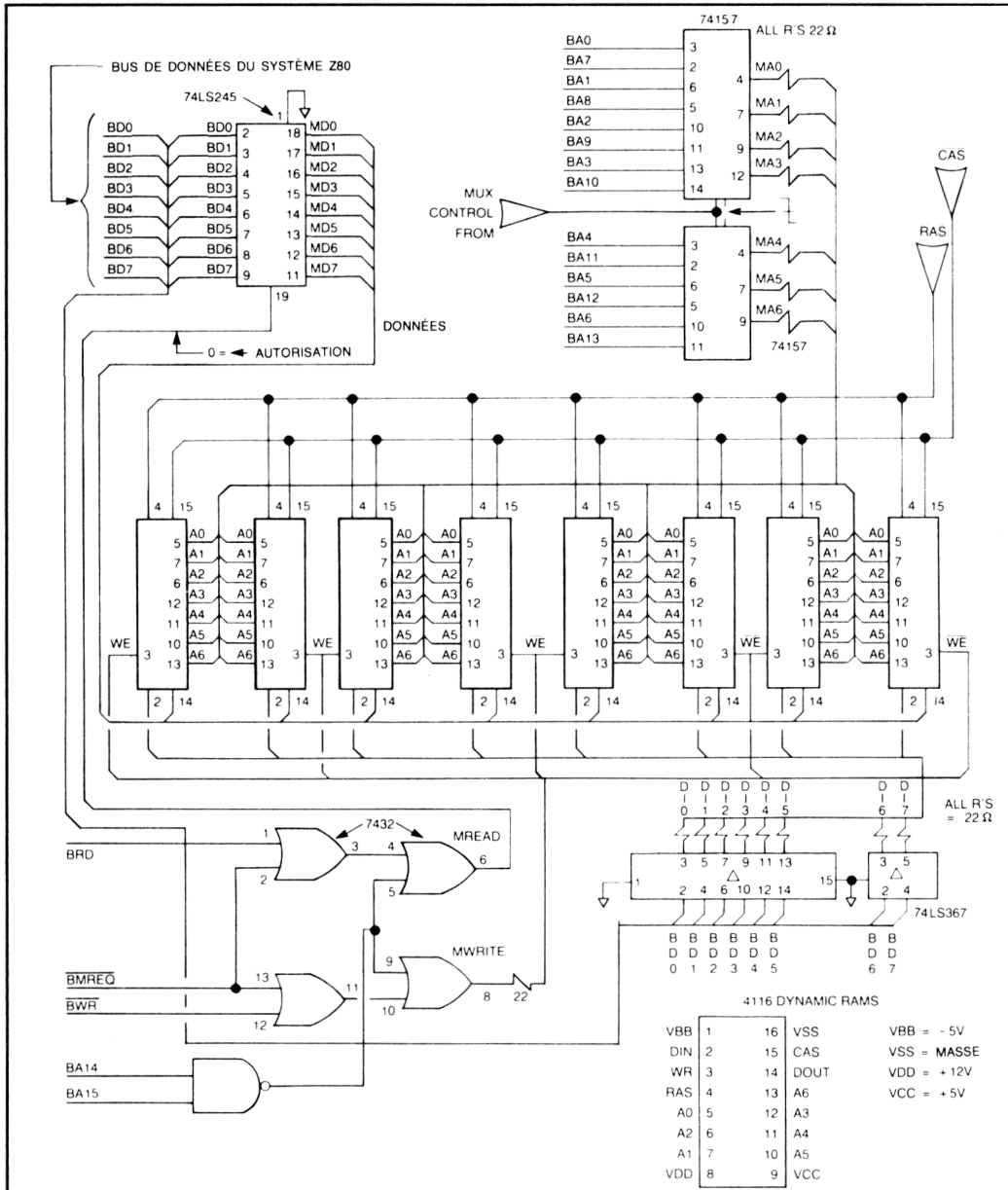


Figure 4.15 : *Schéma complet de mémoire RAM dynamique 16 K × 8 bits interfacée à un microprocesseur Z80.*

d'autres. Dans un système à RAM dynamiques, il faut tenir compte du cas le « pire » qui puisse survenir lorsque l'on calcule le temps entre les adresses de rafraîchissement.

Chaque fois que le Z80 est stoppé (HALT) ou que l'entrée HOLD est activée, il n'y a plus de rafraîchissement des RAM dynamiques. Il en est de même quand on fait une réinitialisation (RESET). Il est important d'être au courant de ces faits quand on envisage l'emploi de RAM dynamiques avec le Z80.

4.9 - Schéma complet d'une RAM dynamique de 16 K × 8 bits

La Figure 4.15 de la page précédente est le schéma complet d'un système à RAM dynamique pouvant être employé avec le Z80. Rappelons-nous, cependant, que la réalisation matérielle de ce système peut être accomplie de plusieurs façons. On a choisi celle-ci parce qu'elle couvre les différents détails essentiels qui doivent être pris en compte lorsque l'on interface le Z80 à un dispositif à RAM dynamique type.

RESUME

Dans ce chapitre, nous avons analysé les notions fondamentales des communications électriques entre Z80 et RAM dynamiques. Nous avons examiné une RAM dynamique type, la 4116. Celle-ci est représentative d'une classe générale de RAM dynamiques qui utilisent le multiplexage pour entrer les adresses.

Nous avons aussi expliqué dans ce chapitre comment effectuer le multiplexage d'adresses en utilisant des circuits réels. Nous avons analysé la fonction des signaux de commande $\overline{RAS}$, MUX et $\overline{CAS}$ pour RAM dynamique et nous avons appris comment élaborer chacun des signaux à partir du microprocesseur Z80.

Enfin, nous avons étudié le rafraîchissement de la RAM dynamique. Cette technique permet d'apprécier l'emploi d'un Z80 avec des RAM dynamiques.

INTERRUPTIONS DU Z80

CHAPITRE 5

INTRODUCTION

Dans ce chapitre, nous allons étudier le sujet général constitué par les interruptions du Z80. Nous allons commencer par présenter la notion d'interruption. Nous montrerons ensuite comment le Z80 réagit électriquement à chaque type différent d'interruption. Nous donnerons, pour chaque mode d'interruption, des exemples qui vous aideront à mieux comprendre ce qui se passe pendant une opération d'interruption.

Il est important de comprendre comment le Z80 gère les demandes d'interruptions externes si vous voulez comprendre les chapitres suivants. Par conséquent, si vous n'êtes pas très familiarisé avec ce sujet, lisez ce chapitre soigneusement.

5.1 - Qu'est-ce qu'une interruption ?

Pour comprendre les interruptions, envisagez la situation suivante. Vous êtes en train de converser avec quelqu'un. Une seconde personne s'approche et vous interpelle, attirant ainsi votre attention. Voici la liste des réactions possibles à cette demande extérieure :

1. Vous pouvez ignorer complètement la seconde personne et continuer votre conversation comme si cette personne n'était pas là.

2. Vous pouvez profiter d'une pause dans votre conversation et tourner alors votre attention vers la seconde personne.

3. Vous pouvez arrêter *immédiatement* votre conversation avec la première personne et vous mettre à parler avec la seconde.

Dans tous les cas, il est probable que lorsque vous aurez fini de parler avec la seconde personne, vous voudrez continuer la conversation que vous aviez avant l'interruption.

Ce scénario peut sembler simpliste, mais il présente bien la notion d'interruption dans un système à microprocesseur.

Pensez-y de la façon que voici. Vous êtes le CPU Z80. La personne avec qui vous étiez en train de parler au début représente le programme principal en cours d'exécution ; la seconde personne représente une demande d'interruption extérieure (c'est-à-dire un circuit quelconque du système qui souhaite attirer l'attention du CPU). Le CPU Z80 doit prendre en compte cette demande extérieure. Il y a plusieurs façons de le faire. Les trois possibilités indiquées précédemment sont les plus répandues.

Après cette introduction générale aux interruptions, examinons maintenant les détails des interruptions du Z80.

5.2 - D'où les demandes d'interruptions proviennent-elles ?

Un système à microprocesseur peut être constitué de nombreux éléments différents. Il peut comporter un tube CRT, une imprimante, une unité à disques souples ou durs, un temporisateur, un moteur de commande, un convertisseur digital-analogique (CDA), pour n'en nommer que quelques-uns. La plupart de ces éléments extérieurs n'ont besoin de l'attention du CPU qu'à certains moments. Aux autres moments, ils fonctionnent de leur côté.

Supposons, par exemple, que votre système ait une horloge comme élément extérieur et que l'horloge doive afficher l'heure sur un écran CRT. Les chiffres de l'horloge sont renouvelés une fois par seconde. En d'autres termes, une fois par seconde les circuits de l'horloge demandent au CPU de lire l'heure et de l'afficher sur l'écran CRT. A tous les autres moments, cependant, le CPU est disponible pour effectuer les autres tâches nécessaires.

Le point principal, ici, est que les circuits extérieurs de l'horloge n'ont pas besoin de l'attention du CPU *pendant la totalité du temps*. Les circuits de l'horloge demandent au CPU de les lire seulement lorsque c'est nécessaire. Ceci peut être accompli par le système d'interruption du CPU par lequel, une fois par seconde, le CPU reçoit une demande d'interruption électrique extérieure provenant de l'horloge. A ce moment, le CPU arrête tout ce qu'il peut être en train de faire et lit l'heure. Après cette lecture et l'affichage de l'heure, le CPU continue le programme qu'il était en train d'exécuter avant l'interruption.

Ceci est un exemple simple d'interruption. Une procédure semblable sera suivie par tous les circuits externes qui émettent vers le CPU une demande d'interruption électrique. La réponse à la question initiale demandant d'où viennent les demandes d'interruptions est donc qu'elles viennent de circuits du système à microprocesseur.

5.3 - Interruptions non masquables

Il y a quatre lignes d'entrée d'interruptions sur le microprocesseur Z80 : $\overline{INT}$, $\overline{NMI}$, $\overline{BUSRQ}$ et $\overline{RES}$. Chacune d'elles peut provoquer une interruption du Z80. La Figure 5.1 montre les quatre lignes d'entrée. Nous allons étudier ici la ligne d'interruption $\overline{NMI}$ ou interruption non masquable. Nous allons commencer par cette interruption parce qu'elle fonctionne d'une seule façon. Mais la plupart des points qui seront analysés s'appliquent aussi bien aux autres types d'entrée d'interruption.

Une *interruption non masquable* est une interruption toujours reconnue par le Z80 aussitôt que c'est électriquement possible. Ce type d'interruption concerne la troisième possibilité de notre énumération de la section 5.1. Comme le Z80 a une interruption non masquable, il s'ensuit qu'il a aussi une interruption masquable (broche d'entrée $\overline{INT}$). (Nous étudierons cette broche d'entrée plus loin.)

La broche 17 du Z80 est référencée $\overline{NMI}$ (interruption non masquable). Cette entrée d'interruption est active sur le front descendant du signal (Voir Figure 5.2). L'entrée $\overline{NMI}$ est complètement asynchrone pour le Z80. Ceci veut dire que les circuits extérieurs peuvent émettre ou activer leur demande à n'importe quel moment sans s'occuper de l'état de fonctionnement du CPU.

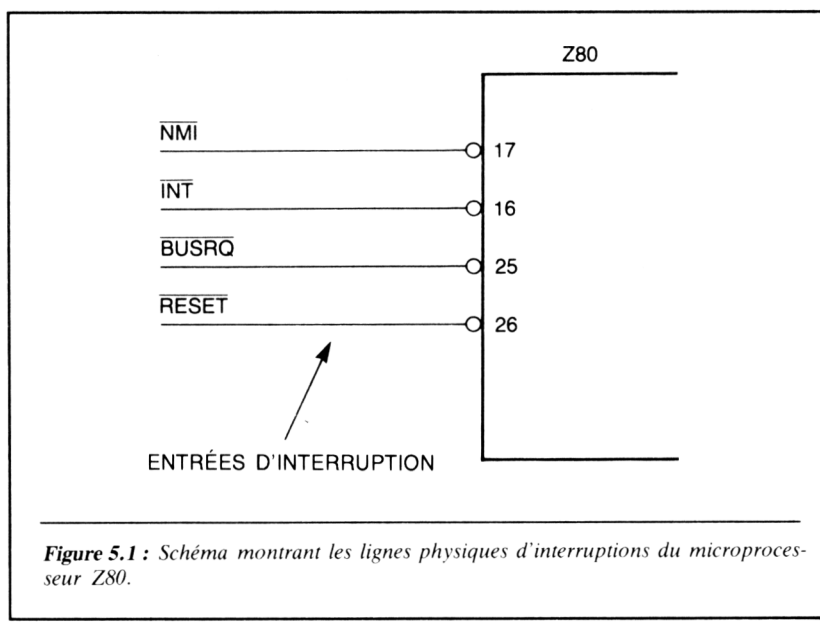


Figure 5.1 : *Schéma montrant les lignes physiques d'interruptions du microprocesseur Z80.*

La question suivante est : « Quand l'entrée d'interruption $\overline{\text{NMI}}$ est-elle reconnue électriquement par le CPU ? » La demande $\overline{\text{NMI}}$ est généralement reconnue par le Z80 à la fin d'un cycle d'instructions en cours. Ceci permet au Z80 de terminer complètement une séquence d'instructions avant toute prise en compte de l'interruption. En permettant au Z80 d'agir de cette façon, la logique interne du CPU peut être simplifiée. De plus, cette séquence ne ralentit pas d'une façon sensible la réponse du CPU aux demandes d'interruption.

La première action qui se passe pendant une opération $\overline{\text{NMI}}$ est que le registre PC (compteur de programme) est empilé. Ceci sauvegarde en mémoire l'adresse à laquelle le Z80 devra retourner après avoir terminé le traitement de l'interruption.

Ensuite, l'état de la bascule d'interruption (IFF1) est stocké en IFF2. IFF1 est référencé « bascule d'autorisation d'interruption » ; sa fonction est d'autoriser ou d'inhiber les interruptions à la broche d'entrée $\overline{\text{INT}}$ du Z80. Elle n'est pas utilisée pour les interruptions non masquables.

IFF1 est stocké en IFF2 parce que le Z80 doit sauvegarder l'état logique de cette bascule pendant la gestion d'une $\overline{\text{NMI}}$ (IFF1 détermine si la demande $\overline{\text{INT}}$ sera acceptée par le CPU). Cette bascule est mise au 0 logique pendant la gestion d'une demande $\overline{\text{NMI}}$. Ceci empêche toutes les interruptions de survenir pendant la gestion d'une NMI. (Plus tard, quand le Z80 revient d'une gestion d'interruption $\overline{\text{NMI}}$, le CPU doit être restauré dans l'état où il était avant la demande $\overline{\text{NMI}}$. Cela veut dire qu'IFF1 doit être mis dans l'état où il était avant l'apparition de l'interruption.)

Après stockage de IFF1 en IFF2 et mise de IFF1 au 0 logique, le Z80 saute à l'emplacement mémoire 0066H.

Résumons maintenant rapidement les quatre événements qui ont lieu dans le Z80 quand une $\overline{\text{NMI}}$ est acceptée. En bref :

Figure 5.2 : *La ligne d'entrée d'interruption $\overline{\text{NMI}}$ est active sur le front descendant du signal. Quand elle est activée, elle met au 1 logique un latch interne qui garde en mémoire la demande d'interruption.*

1. Le PC est empilé.

2. IFF1 est stocké en IFF2.

3. IFF1 est mis au 0 logique.

4. Le Z80 saute à l'emplacement 0066H.

L'emplacement 0066H contient la routine de gestion qui doit être exécutée pendant une $\overline{\text{NMI}}$. (Notez que les registres du CPU ne sont pas tous sauvegardés quand une interruption survient. Le programmeur doit les sauvegarder, s'il le désire, pendant la routine de gestion d'interruption.) Un début type de routine de gestion d'interruption est montré en Figure 5.3.

Une autre solution pour sauvegarder tous les registres est d'employer les instructions d'échange de registres. Ceci permet d'utiliser l'autre jeu de registres pendant l'exécution d'une interruption. La Figure 5.4 montre les instructions qui permettent de le faire.

```
EMPILER AF
EMPILER BC   } SAUVEGARDER TOUS LES REGISTRES
EMPILER DE
EMPILER HL
```

Figure 5.3 : *Un début type de routine d'interruption sauvegardant l'état des registres internes. Ce n'est pas toujours nécessaire et cela dépend de l'application du système global.*

```
EX    AF, AF'  } SAUVEGARDER LES REGISTRES
EXX
```

(NOTA : CES DEUX INSTRUCTIONS VONT SAUVEGARDER LES CONTENUS DE TOUS LES REGISTRES DU CPU.)

Figure 5.4 : *Autre méthode pour sauvegarder les registres internes du Z80.*

5.4 - Effacement de la demande $\overline{\text{NMI}}$

Le signal électrique de demande $\overline{\text{NMI}}$ a été produit par un circuit extérieur au Z80. Le Z80 a une action à effectuer : effacer la condition (si c'est possible ou nécessaire) qui a causé l'interruption. Une façon d'effacer la demande d'interruption est d'écrire dans un port de sortie (nous verrons plus tard une autre manière d'y parvenir). Il est essentiel que le programmeur supprime électriquement la demande d'interruption.

5.5 - Fin de la routine de gestion de $\overline{\text{NMI}}$

A la fin de la routine de gestion de $\overline{\text{NMI}}$, le Z80 exécute l'instruction RETN (retour d'interruption non masquable). A ce moment-là, les deux octets de données du haut de la pile sont utilisés pour l'adresse de retour. IFF2 est recopié dans IFF1. (*Note :* On peut utiliser l'instruction RET pour cette opération, mais IFF2 ne sera pas recopié dans IFF1.)

5.6 - Exemple de fonctionnement avec $\overline{\text{NMI}}$

Voici un exemple qui illustre l'utilisation de l'interruption non masquable. Nous allons montrer une technique que l'on peut utiliser pour émettre une demande $\overline{\text{NMI}}$ et nous allons présenter le logiciel nécessaire pour gérer l'interruption. La Figure 5.5 montre un circuit qui peut être employé par une $\overline{\text{NMI}}$. Examinons-le.

Quand le poussoir S1 est enfoncé (Voir Figure 5.5), la broche 3 du boîtier 7400 passe au 0 logique. Ceci met au 0 logique l'entrée d'horloge (C) de la bascule type D. Rien ne se passe à ce moment-là. Quand S1 est relâché, l'entrée d'horloge de la bascule type D va au 1 logique. A ce moment-là, la sortie $\overline{\text{Q}}$ est mise au 0 logique. $\overline{\text{NMI}}$ est demandé électriquement à la broche d'entrée 17 du Z80.

L'entrée $\overline{\text{NMI}}$ va rester au 0 logique jusqu'à ce que le Z80 effectue une opération d'écriture de port 0FFH. La sortie $\overline{\text{Q}}$ de la bascule type D est alors mise au 1 logique. Ceci entraîne la mise au 1 logique de l'entrée $\overline{\text{NMI}}$ du Z80.

Il faut noter ici que l'entrée $\overline{\text{NMI}}$ est sensible à un front et non pas à un niveau. Autrement dit, lorsque l'entrée $\overline{\text{NMI}}$ passe de 1 à 0, la demande d'interruption est « latchée » par le Z80 (Voir Figure 5.6).

A cause des caractéristiques de déclenchement par front de l'entrée $\overline{\text{NMI}}$, on aurait pu employer un circuit de demande d'interruption très simple,

comme celui de la Figure 5.7. On voit que, lorsque le contact est appuyé, $\overline{NMI}$ est accepté par le Z80. Il faut ensuite relâcher le contact, puis le presser à nouveau avant qu'une autre interruption ne se produise. Notez que, contrairement au circuit de la Figure 5.5, le Z80 n'aura pas à ôter la demande d'interruption dans ce circuit.

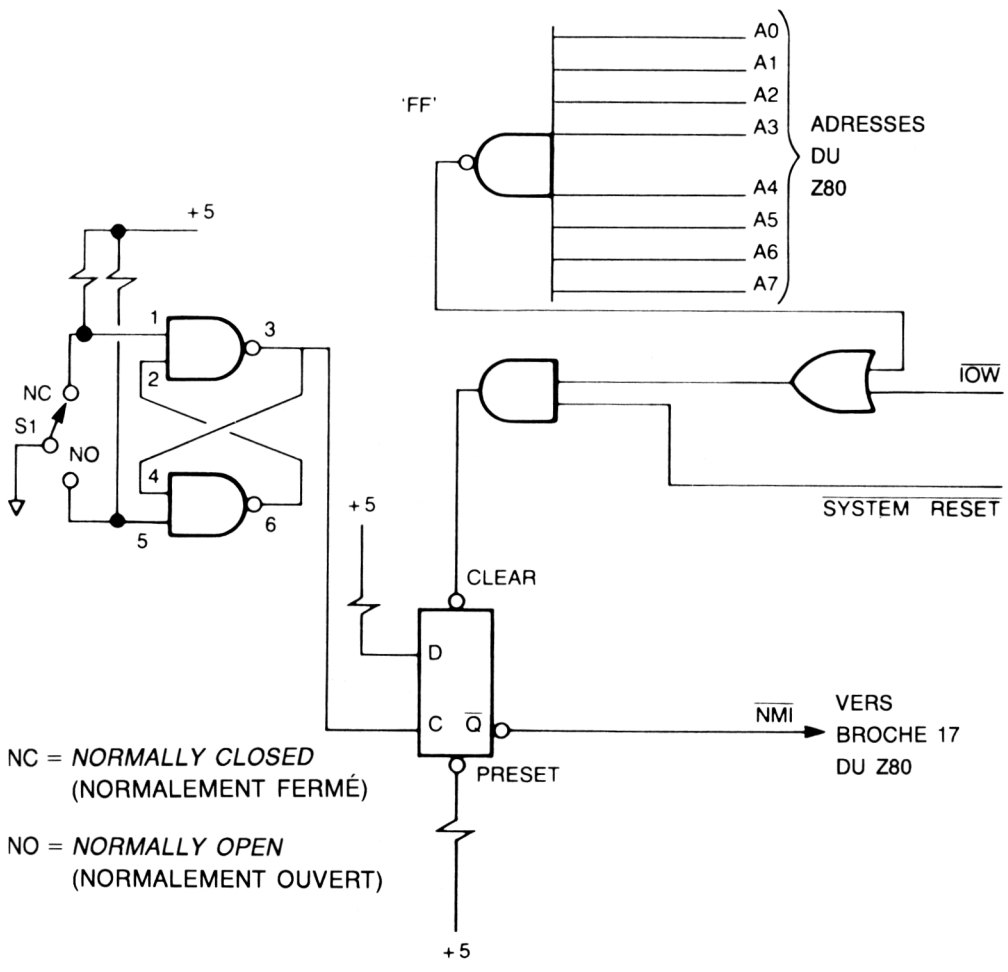


Figure 5.5 : *Schéma montrant une façon de produire une demande $\overline{NMI}$ par un circuit extérieur.*

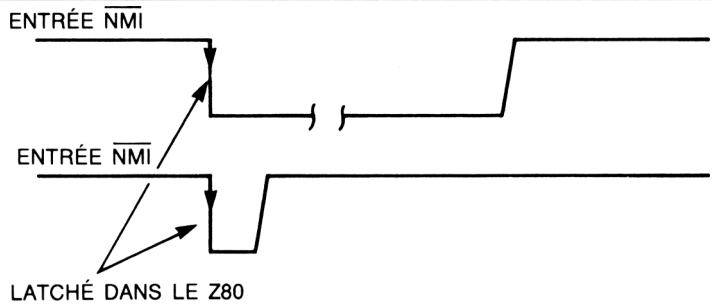


Figure 5.6 : *Diagramme des temps montrant comment la demande $\overline{NMI}$ peut être ôtée de la ligne d'entrée avant que l'interruption ne soit prise en compte électriquement par le microprocesseur du système.*

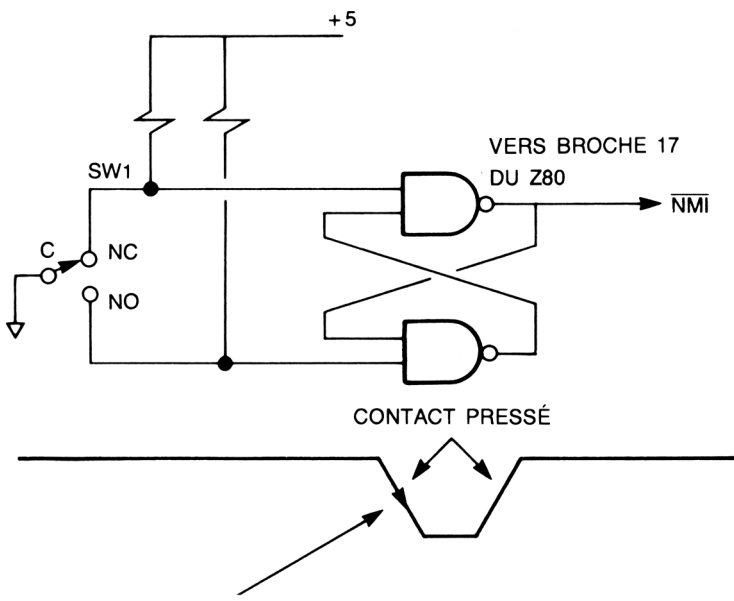


Figure 5.7 : *Méthode simple pour produire une demande $\overline{NMI}$ externe pour Z80.*

5.7 - Résumé des opérations $\overline{\text{NMI}}$

L'analyse qui précède a porté sur les points importants concernant l'entrée $\overline{\text{NMI}}$ du Z80. Passons-les en revue :

1. $\overline{\text{NMI}}$ est non masquable, c'est-à-dire qu'elle sera toujours acceptée.
2. $\overline{\text{NMI}}$ est déclenchée par front sur une transition de 1 à 0.
3. Le Z80 empile le PC.
4. IFF1 est stocké en IFF2, ce qui sauvegarde l'état logique de la bascule d'interruption.
5. IFF1 est mis au 0 logique, ce qui inhibe d'autres interruptions.
6. Le Z80 saute à l'adresse mémoire 0066H et commence à exécuter la routine.
7. RETN provoque le retour de l'adresse du PC et restaure la condition logique de la bascule d'autorisation d'interruption.

5.8 - Entrée $\overline{\text{INT}}$

Nous allons examiner maintenant comment agit la broche d'entrée 16, $\overline{\text{INT}}$, du microprocesseur Z80. Une grande partie des circuits utilisés par $\overline{\text{NMI}}$ servent aussi à $\overline{\text{INT}}$.

Une caractéristique électrique de l'entrée $\overline{\text{INT}}$ du Z80 est qu'elle peut être masquée logiquement, c'est-à-dire qu'il y a une instruction du Z80, qui s'appelle DI (*Disable Interrupts*, inhiber les interruptions) et qui, lorsqu'elle est exécutée, fait une coupure logique entre l'entrée $\overline{\text{INT}}$ et le microprocesseur Z80. Rappelez-vous que IFF1 est la bascule interne du Z80 qui valide ou inhibe l'entrée $\overline{\text{INT}}$. Elle est mise au 0 logique.

Bien que nous ayons examiné IFF1 précédemment, nous n'avons pas détaillé son emploi. Contrairement à $\overline{\text{NMI}}$, l'entrée $\overline{\text{INT}}$ est sensible à un niveau (et non sensible à une transition) ; ce qui signifie que l'entrée $\overline{\text{INT}}$ doit rester au 0 logique jusqu'à ce que le Z80 la prenne en compte. La prise en compte est effectuée à la fin du dernier cycle machine d'un cycle d'instructions. Rappelez-vous que l'entrée $\overline{\text{NMI}}$ pouvait être coupée du Z80 avant sa prise en compte. Nous connaissons maintenant deux points importants concernant l'entrée $\overline{\text{INT}}$ du Z80 :

1. $\overline{\text{INT}}$ peut être masquée électriquement par le Z80, en employant l'instruction DI.

2. L'entrée $\overline{\text{INT}}$ doit rester à un 0 logique valide jusqu'à sa prise en compte par le Z80. L'entrée $\overline{\text{INT}}$ est sensible au niveau, et non pas à une transition comme $\overline{\text{NMI}}$. Ceci signifie que l'entrée $\overline{\text{INT}}$ *doit* être ôtée après que l'interruption ait été traitée. Sinon une autre demande d'interruption sera accordée. C'est au logiciel du système de supprimer la demande d'interruption afin d'éviter une autre interruption non désirée.

Certains dispositifs périphériques prévus pour utilisation avec le Z80 coupent automatiquement la demande d'interruption au moment voulu. Un exemple de ce type de dispositif est le PIO Z80. Le PIO reconnaît le moment où le Z80 est en train d'exécuter une instruction RETI et coupe automatiquement la demande sur la ligne $\overline{\text{INT}}$ du système. Nous analyserons le PIO et d'autres dispositifs en détail dans les chapitres suivants.

Il n'y a qu'une seule broche d'entrée $\overline{\text{INT}}$ sur le Z80, mais elle fonctionne de trois façons (modes). Quand on utilise la structure d'interruption du Z80, on doit programmer le mode qui sera employé. Ceci est effectué avant toute entrée d'interruption dans le CPU. Les noms des modes pour les interruptions du Z80 sont référencés mode 0, mode 1, et mode 2. Pour programmer ces trois modes, il y a trois instructions distinctes : IM0, IM1, IM2.

Quand le Z80 est mis sous tension ou remis à zéro, il est en mode d'interruption 0. En outre, la bascule d'interruption IFF1 est mise au 0 logique et les interruptions sont interdites. Examinons maintenant, l'un après l'autre, les trois modes d'opération d'interruption, et voyons leurs différences et leurs ressemblances.

5.9 - Interruption en mode 1

Commençons par les interruptions en mode 1. Nous débuterons par ce mode parce que c'est celui qui ressemble le plus à $\overline{\text{NMI}}$, ce qui nous permettra de partir sur un terrain familier. Le Z80 peut être programmé en mode 1 en employant l'instruction IM1.

Nous supposerons ici que les interruptions ont été autorisées par l'instruction EI (*Enable Interrupt*, autoriser interruptions) et que le Z80 est prêt à traiter les demandes d'interruptions qui parviennent sur la ligne d'entrée $\overline{\text{INT}}$.

Quand l'entrée $\overline{\text{INT}}$ va au 0 logique et que le Z80 accuse réception de l'entrée, le processus suivant a lieu :

1. IFF1 est mis au 0 logique, interdisant ainsi toute autre interruption provenant de la ligne $\overline{\text{INT}}$.

2. IFF2 est mis au 0 logique. Rappelez-vous que pour $\overline{\text{NMI}}$, IFF2 servait d'emplacement de stockage temporaire pour IFF1 ; mais dans ce mode d'interruption, IFF2 n'est pas employé dans ce but.

3. Le PC est stocké dans la pile du programme. Ce sera l'adresse mémoire de l'instruction à exécuter quand la gestion d'interruption sera terminée.

4. Le Z80 saute à l'emplacement 0038H de la mémoire du système et commence à exécuter les instructions du logiciel.

On peut voir d'après cette énumération que l'entrée d'interruption $\overline{\text{NMI}}$ effectue essentiellement les mêmes types d'opérations.

Pendant l'exécution de la routine d'interruption, les registres du CPU doivent être sauvegardés, si la routine les utilise. Ceci peut être effectué en employant l'instruction EX (*EXchange*, échange), ou en empilant les registres utilisés dans la pile du système.

Pendant l'exécution de cette routine, la demande d'interruption doit être logiquement séparée de la broche 16 du Z80. En fait, le logiciel destiné à accomplir cette tâche varie d'un système à l'autre suivant les circuits employés pour activer la demande d'interruption. La Figure 5.5 montre une façon permettant à des circuits d'activer l'entrée de demande d'interruption. On voit dans cette figure que le Z80 effectue une opération d'écriture dans le port 0FFH afin d'ôter la demande d'interruption de la broche 16 d'entrée $\overline{\text{INT}}$. Pour que le Z80 accepte d'autres interruptions, la bascule IFF1 doit être mise au 1 logique. C'est ce qui est effectué quand le CPU exécute l'instruction EI avant de recommencer une routine provoquée par une interruption.

Le programmeur doit, enfin, permettre au CPU de reprendre l'exécution du programme qui se déroulait avant que la demande d'interruption ne soit activée. Ceci est réalisé par l'emploi de l'instruction RET ou RETI. L'instruction RET retire les deux octets du sommet de la pile système et les stocke dans le PC du Z80. Rappelez-vous que ces deux octets sont l'adresse de l'instruction que le CPU aurait exécutée s'il n'y avait pas eu d'interruption. C'est la même instruction RET qui est exécutée à la fin d'un sous-programme faisant suite à un appel de sous-programme.

L'instruction RETI effectue la même opération que l'instruction RET. La différence essentielle est que certains périphériques, tels que le PIO du Z80, sont prévus pour reconnaître électriquement le code opération RETI. Quand un dispositif extérieur reconnaît ce code opération, la demande d'interruption est renouvelée. (Vous en apprendrez davantage à ce sujet dans les chapitres suivants.)

La Figure 5.8 présente un exemple de routine d'interruption. On suppose ici que le Z80 a été programmé en mode 1 et que l'interruption a

été appliquée au Z80 en employant les circuits de la Figure 5.5. Dans la Figure 5.8, l'instruction OUT(0FFH),A sert à effacer la demande d'interruption avant que les interruptions du système ne soient à nouveau autorisées par l'instruction EI.

On doit remarquer que les interruptions ne sont pas autorisées aussi longtemps que le code opération qui suit l'instruction EI n'est pas exécuté. Ceci permet au Z80 d'exécuter l'instruction de retour à la fin de la routine d'interruption — avant qu'une autre demande d'interruption ne soit acceptée d'une façon interne.

Dans le programme de la Figure 5.8 les interruptions sont inhibées dès que le Z80 accepte la demande d'interruption. Ceci se produit quand le Z80 met IFF1 au 0 logique. IFF1 va rester au 0 logique jusqu'à ce qu'il soit à nouveau mis à 1 par le logiciel du système. Ceci peut être effectué à la fin de n'importe quelle routine d'interruption qui n'autorise pas d'interruption

```
ORG     0038H

PUSH AF  }
PUSH HL  }  SAUVEGARDER TOUS LES REGISTRES OU
PUSH DE  }  SEULEMENT CEUX UTILISÉS DANS LA
PUSH BC  }  ROUTINE.

(ROUTINE DE SERVICE)
        SECTION


POP BC  }
POP DE  }  RESTAURER TOUS LES REGISTRES
POP HL  }
POP AF  }

OUT (0FFH),A                  METTRE A 0 INTERRUPTION
      EI                      AUTORISER INTERRUPTIONS
RET OR RETI
```

Figure 5.8 : *Exemple de routine d'interruption qui suppose que le Z80 a été programmé en structure d'interruption mode 1.*

pendant qu'une interruption est gérée. Le programmeur, cependant, peut choisir d'autoriser les interruptions pendant la routine pour permettre d'entrer d'autres interruptions. C'est à l'auteur du système de décider si une routine d'interruption doit pouvoir être interrompue par une autre demande.

5.10 - Interruption en mode 0

Nous allons examiner ici un second mode de fonctionnement de la structure d'interruption du Z80 : le mode 0. Ce mode est autorisé quand le CPU est mis sous tension, quand la commande de remise à 0 est activée ou quand une instruction IM0 est exécutée. Cette sorte d'interruption est souvent comparée à celle du 8080.

Avec l'interruption en mode 0, le CPU entre une demande d'interruption de la même manière qu'en mode 1 (décrite ci-dessus). Cependant, plutôt que de faire sauter le CPU à l'emplacement 0038H seulement, on peut le faire sauter à l'une des huit adresses présélectionnées ou, dans ce cas, à n'importe quelle adresse mémoire. C'est une caractéristique très intéressante. Examinons-en le fonctionnement.

En supposant que la demande d'interruption ait été appliquée au Z80 et qu'elle soit acceptée, voici ce qui se passe :

1. IFF1 et IFF2 sont mis au 0 logique, inhibant ainsi toute autre demande d'interruption.

2. Les lignes de sortie $\overline{M1}$ et $\overline{IORQ}$ du Z80 vont toutes les deux au 0 logique. Ceci n'a lieu, pour le Z80, que pendant une interruption et s'appelle état de *reconnaissance d'interruption.*

3. Des circuits extérieurs décodent $\overline{M1}$ et $\overline{IORQ}$ à 0 et valident un octet unique dans le bus de données du Z80. Cet octet est le code opération pour un RST 0-RST 7. Il pourrait être aussi une instruction CALL. Nous supposerons ici que l'octet placé dans le bus de données du système est une instruction RST.

4. Le Z80 lit l'octet et l'interprète comme un code opération.

5. Le PC est empilé dans la pile du système.

6. Le Z80 saute, enfin, à l'emplacement mémoire spécifié par l'instruction RST.

Ces emplacements comportent :

RST	OCTET DE DONNEES	EMPLACEMENT MEMOIRE (Hex)
0	C7	0000
1	CF	0008
2	D7	0010
3	DF	0018
4	E7	0020
5	EF	0028
6	F7	0030
7	FF	0038

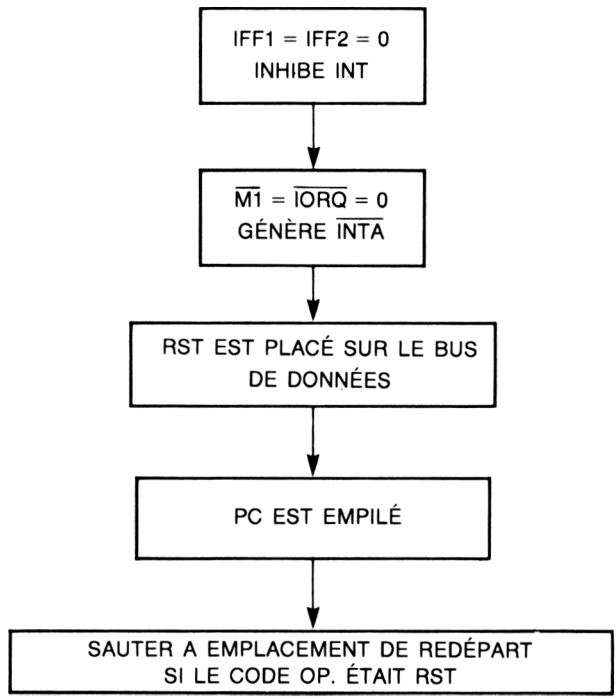


Figure 5.9 : *Ordinogramme montrant le déroulement d'opérations internes qui ont lieu lorsque le Z80 reçoit électriquement une demande d'interruption en mode 0.*

La Figure 5.9 montre une séquence d'opérations des circuits pendant ce fonctionnement. La Figure 5.10 montre une façon de réaliser cette interruption particulière. Voici comment fonctionne ce circuit.

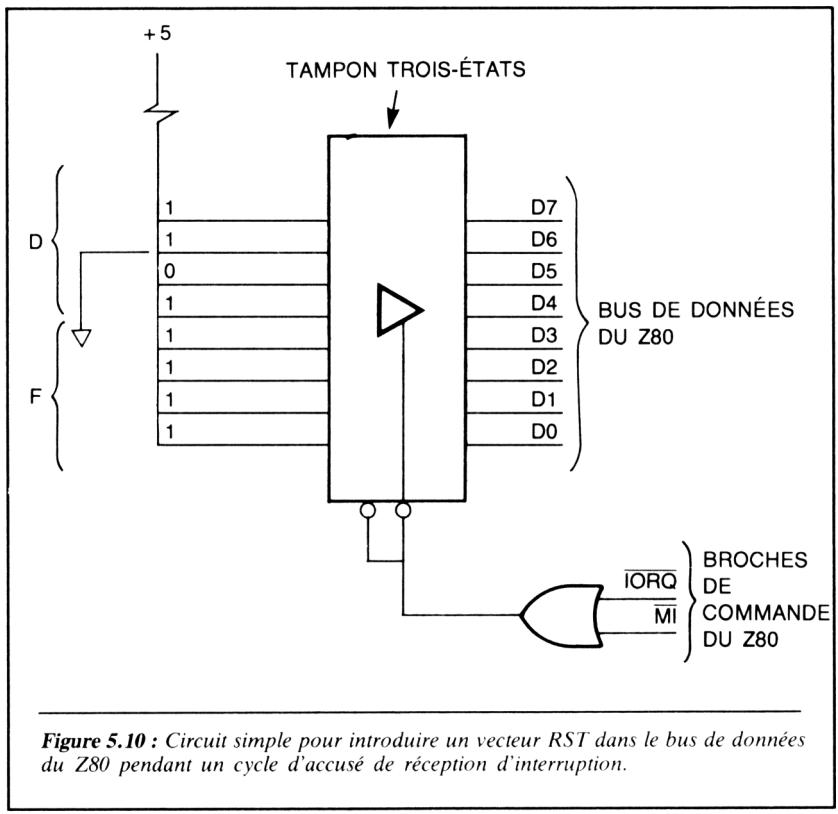


Figure 5.10 : *Circuit simple pour introduire un vecteur RST dans le bus de données du Z80 pendant un cycle d'accusé de réception d'interruption.*

$\overline{M1}$ et $\overline{IORQ}$ sont appliqués à un opérateur logique OU. Quand la demande d'interruption est reçue par le Z80 et acceptée, $\overline{M1}$ et $\overline{IORQ}$ vont au 0 logique et produisent ainsi un accusé de réception d'interruption. La sortie du OU est alors au 0 logique. Dans cet exemple, le 0 logique valide le tampon à 3 états et met un DF (hexadécimal) dans le bus de données du système. Un DF est égal à une instruction RST 3. Il conduit le Z80 à exécuter un CALL à l'adresse mémoire 0018H.

Cet exemple a expliqué une façon de mettre un octet de données dans le bus de données du système. Il a utilisé tout ce qui sert à une interruption en mode 0. Il ne peut, cependant, manipuler qu'un seul dispositif extérieur demandant une interruption. Nous verrons plus loin comment le Z80 peut s'occuper d'une demande d'interruption multiple. Pour l'instant, il est important que vous compreniez comment le CPU gère les interruptions en mode 0.

Nous avons dit au début de ce paragraphe qu'il est possible de mettre une instruction CALL plutôt qu'une instruction RST sur le bus de données.

Quand le Z80 a le code opération du CALL sur le bus pendant le temps de reconnaissance d'interruption, il s'attend à recevoir les deux octets suivants pour l'adresse du CALL.

Les deux cycles machine suivant l'accusé de réception d'interruption sont des cycles de lecture en mémoire. Pendant ces deux cycles, l'adresse de l'appel est lue par le CPU. C'est au circuit extérieur à mettre l'adresse dans le bus au moment approprié. L'ordinogramme de la Figure 5.11 montre

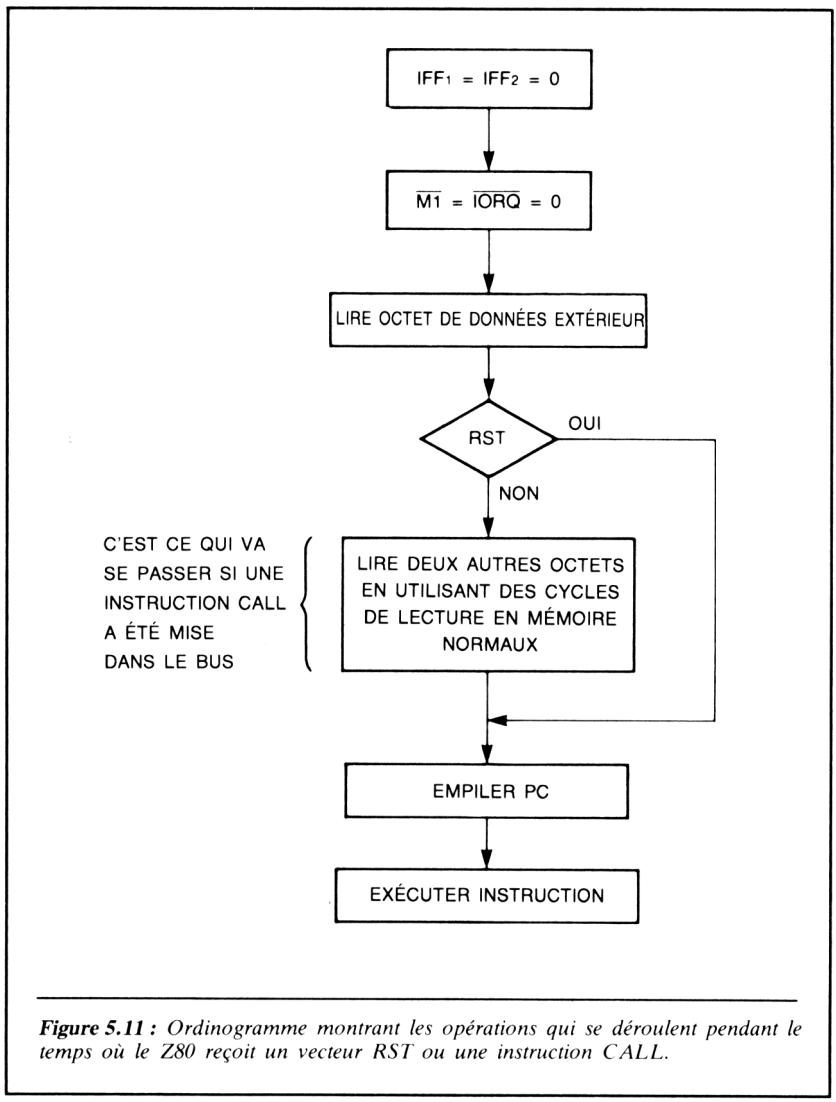


Figure 5.11 : *Ordinogramme montrant les opérations qui se déroulent pendant le temps où le Z80 reçoit un vecteur RST ou une instruction CALL.*

exactement ce qui se passe dans le bus du Z80 lorsque, soit l'instruction RST, soit l'instruction CALL, est mise dans le bus de données du système pendant le cycle d'accusé de réception d'interruption.

La Figure 5.11 montre que l'instruction RST s'exécute plus vite que l'instruction CALL, mais cette dernière présente l'avantage de permettre au Z80 de sauter à n'importe quel emplacement pour exécuter la routine de service d'interruption.

5.11 - Interruption en mode 2

Nous allons étudier le troisième mode d'interruption du Z80, le mode 2. Ce mode est très puissant et simple à utiliser. Les interruptions en mode 2 sont établies par le microprocesseur Z80 quand il exécute l'instruction IM2. Ces interruptions sont demandées électriquement exactement de la même façon que des interruptions en mode 0 et en mode 1. Nous allons donc maintenant nous pencher sur la façon par laquelle le Z80 répond à une demande d'interruption en mode 2.

Le mode 2 a la structure générale suivante. Quand le Z80 reconnaît la demande d'interruption par l'activation simultanée de $\overline{M1}$ et $\overline{IORQ}$, les circuits mettent un octet de données dans le bus de données du système. C'est ce qui se passe aussi lors d'un accusé de réception d'interruption en mode 0 ; mais la similitude se termine là. Le Z80 prend l'octet de données et forme un mot de 16 bits avec un registre interne à 8 bits. Ce registre interne est le registre I. (Voir Figure 5.12.)

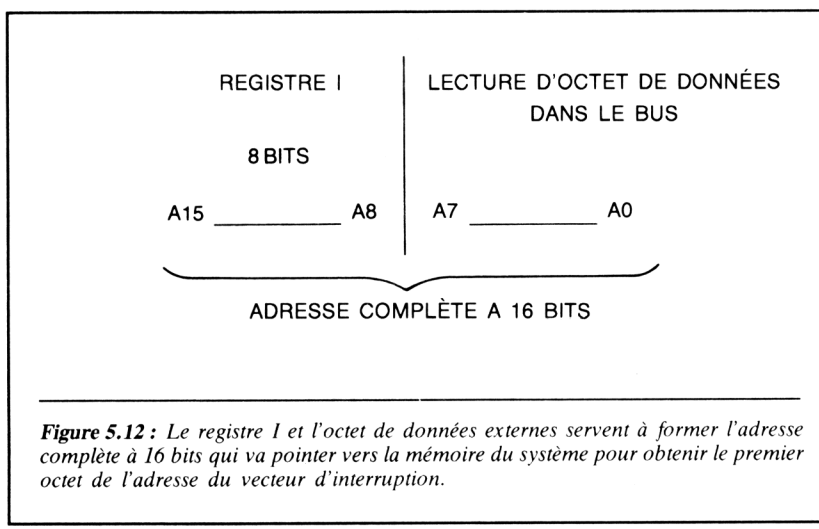


Figure 5.12 : *Le registre I et l'octet de données externes servent à former l'adresse complète à 16 bits qui va pointer vers la mémoire du système pour obtenir le premier octet de l'adresse du vecteur d'interruption.*

Les données situées à la nouvelle adresse à 16 bits constituent une autre adresse. Cette nouvelle adresse est l'adresse mémoire absolue de la routine de gestion d'interruption (Voir Figure 5.13). Généralement c'est le fonctionnement des interruptions en mode 2. Examinons un exemple.

Nous supposerons dans cet exemple que lorsque l'interruption en mode 2 fait l'objet d'un accusé de réception, le dispositif extérieur met 52H dans le bus de données. Nous supposerons, en outre, que le registre I du Z80 a été initialisé à 19H et que les emplacements mémoire 1952H et 1953H contiennent :

```
1952H = 38H en données
1953H = 2FH en données
```

Supposons que l'interruption vienne de se produire. 52H du bus de données et 19H du registre I sont groupés ensemble, donnant ainsi une adresse mémoire 1952H. Le registre I prend les 8 bits de poids fort et

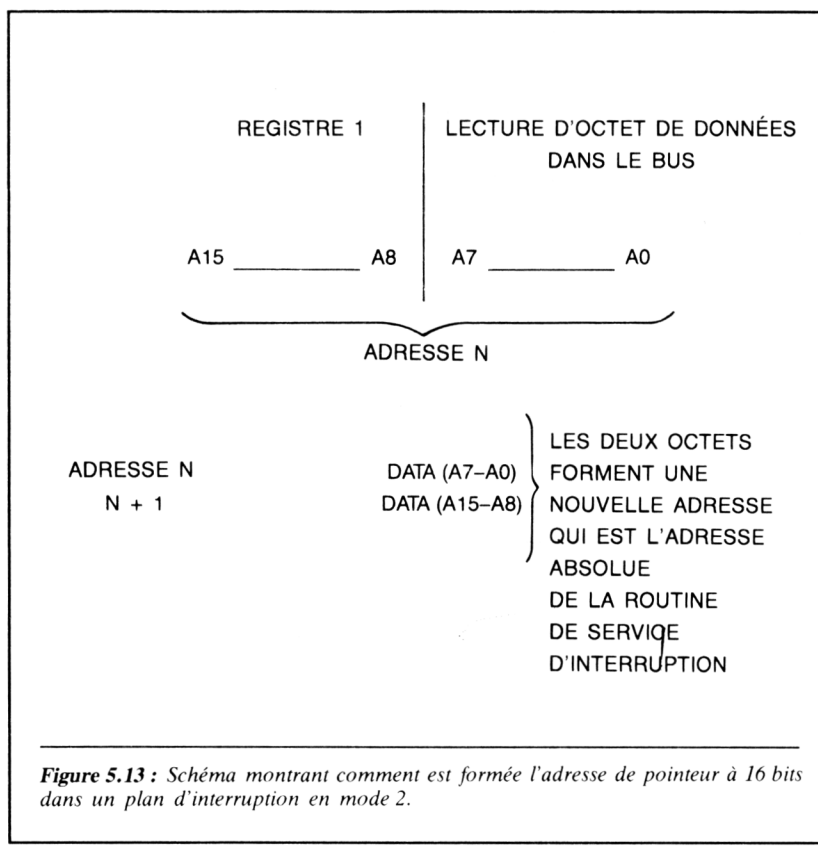


Figure 5.13 : *Schéma montrant comment est formée l'adresse de pointeur à 16 bits dans un plan d'interruption en mode 2.*

l'octet de données lu dedans prend les 8 bits de poids faible. Le Z80 prend les données stockées à cette adresse et les utilise comme octet de poids faible d'une nouvelle adresse mémoire. Dans cet exemple, l'octet de poids faible de la nouvelle adresse serait 38H. Le Z80 lit ensuite l'octet de données à l'adresse mémoire 1952H + 1. Cette donnée sert d'octet de poids fort pour la nouvelle adresse mémoire. L'adresse mémoire totale formée est 2F38H. Le Z80 saute maintenant à l'adresse et exécute la routine de gestion d'interruption.

Ceci peut sembler nécessiter beaucoup d'efforts pour former une simple adresse d'interruption, mais examinons ce que ce type de structure donne dans un système. Supposez que vous ayez un système comportant plusieurs (disons 10) dispositifs extérieurs qui peuvent requérir une interruption. On peut s'arranger pour que chacun de ces dispositifs place un octet différent sur le bus de données quand l'interruption est acquittée. En lisant cet octet distinct on peut faire sauter le Z80 à n'importe quelle adresse mémoire du système. De plus, le Z80 peut aller à une adresse différente pour chacun des 10 dispositifs. Voyons maintenant un exemple montrant comment cela peut se passer dans un système.

Pour commencer, il faut connaître l'adresse du système où réside la routine de gestion d'interruption. Nous supposerons, dans notre cas, que nous avons les adresses suivantes :

DISPOSITIF	ADRESSE MEMOIRE
DEV1	20F5H
DEV2	3802H
DEV3	1951H
DEV4	F318H
DEV5	E821H
DEV6	2568H
DEV7	1585H
DEV8	CE80H
DEV9	3597H
DEV10	211EH

Nous devons maintenant construire une table des vecteurs qui seront les adresses mémoire qui pointent vers l'adresse absolue de la routine de gestion. Dans l'exemple précédent, 2F38H était un vecteur pointant vers la routine de service d'un composant spécifique.

Les routines de gestion d'interruption qui précèdent vont résider à ces emplacements spécifiés dans la mémoire du système. (*Note :* Dans cet exemple, les emplacements ont été choisis arbitrairement et servent seulement à illustrer les notions que nous sommes en train d'étudier.)

Ensuite, nous devons décider où, dans la mémoire du système, placer la table des vecteurs d'adresses. Il y a un maximum de 256 emplacements mémoire servant aux vecteurs d'interruptions. Chaque vecteur nécessite deux octets ; donc, un maximum de 128 adresses mémoire distinctes ou vecteurs doit être spécifié lors d'interruptions en mode 2. Cependant, c'est rarement une limitation dans la conception du système.

Dans cet exemple, nous établirons notre espace de vecteurs aux emplacements 1100H-11FFH. C'est un choix arbitraire (il est généralement déterminé par le concepteur du système). Une fois cet espace établi, nous

```
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   PROGRAMME POUR ETABLIR UNE INTERRUPTION EN MODE 2 POUR Z80
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
0000                    CODE 0000
;
0000   ED5E             IM 2        ;ETABLIR MODE 2 D'INTERR.
0002   3E11             LD A,11H    ;OCTET DE POIDS FORT DE VECTEUR ADDR
0004   ED47             LD I,A
0006   FB               EI          ;AUTORISER INTERRUPTIONS
;
;
;  TOUTES LES ROUTINES DE GESTION SERONT ECRITES
;  A CET EMPLACEMENT DU LISTING.
;  ELLES NE SERONT PAS MONTREES.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; POUR ETABLIR MAINTENANT LA TABLE DES VECTEURS
; CE QUI SUIT N'EST QU'UNE FAÇON DE LE FAIRE.
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
1100                    CODE 1100H     ;ADRESSE DE LA TABLE DES VECTEURS
;
1100   F520             DEFW 20F5H     ;ORGANE 1
1102   0238             DEFW 3802H     ;ORGANE 2
1104   5119             DEFW 1951H     ;ORGANE 3
1106   18F3             DEFW 0F318H    ;ORGANE 4
1108   21E8             DEFW 0E821H    ;ORGANE 5
110A   6825             DEFW 2568H     ;ORGANE 6
110C   8515             DEFW 1585H     ;ORGANE 7
110E   80CE             DEFW 0CE80H    ;ORGANE 8
1110   9735             DEFW 3597H     ;ORGANE 9
1112   1E21             DEFW 211EH     ;ORGANE 10
;
;   FIN DE LA TABLE DES VECTEURS
```

Figure 5.14 : *Echantillon de programme Z80 configurant le microprocesseur pour des interruptions en mode 2 et générant la table des vecteurs.*

devons décider quel octet de données le dispositif extérieur mettra sur le bus de données quand l'interruption sera acquittée par le Z80. Pour le moment, supposons que les dispositifs suivent le plan suivant :

DISPOSITIF	OCTET MIS DANS LE BUS DE DONNEES
DEV1	00H
DEV2	02H
DEV3	04H
DEV4	06H
DEV5	08H
DEV6	0AH
DEV7	0CH
DEV8	0EH
DEV9	10H
DEV10	12H

Remarquez que chaque octet va mettre un nombre pair dans le bus de données. C'est parce que le vecteur total va occuper deux emplacements consécutifs dans la mémoire du système. Une fois ce renseignement connu, nous pouvons remplir la table des vecteurs. Dans cet exemple, elle apparaîtrait comme ceci :

ADRESSE MEMOIRE	DONNEES	VECTEUR DE DISPOSITIF
1100H	F5H	DEV1
1101H	20H	
1102H	62H	DEV2
1103H	38H	
1104H	51H	DEV3
1105H	19H	
1106H	18H	DEV4
1107H	F3H	
1108H	21H	DEV5
1109H	E8H	
110AH	68H	DEV6
110BH	25H	
110CH	85H	DEV7
110DH	15H	
110EH	80H	DEV8
110FH	CEH	
1110H	97H	DEV9
1111H	35H	
1112H	1EH	DEV10
1113H	21H	

Un programme de Z80 qui établit cet exemple apparaît dans la Figure 5.14.

5.12 - Interruptions demandées par plusieurs dispositifs

Nous allons montrer comment plusieurs dispositifs peuvent demander une interruption au Z80. Nous utiliserons cela plus loin lorsque nous étudierons les dispositifs périphériques du Z80. La Figure 5.15 montre le

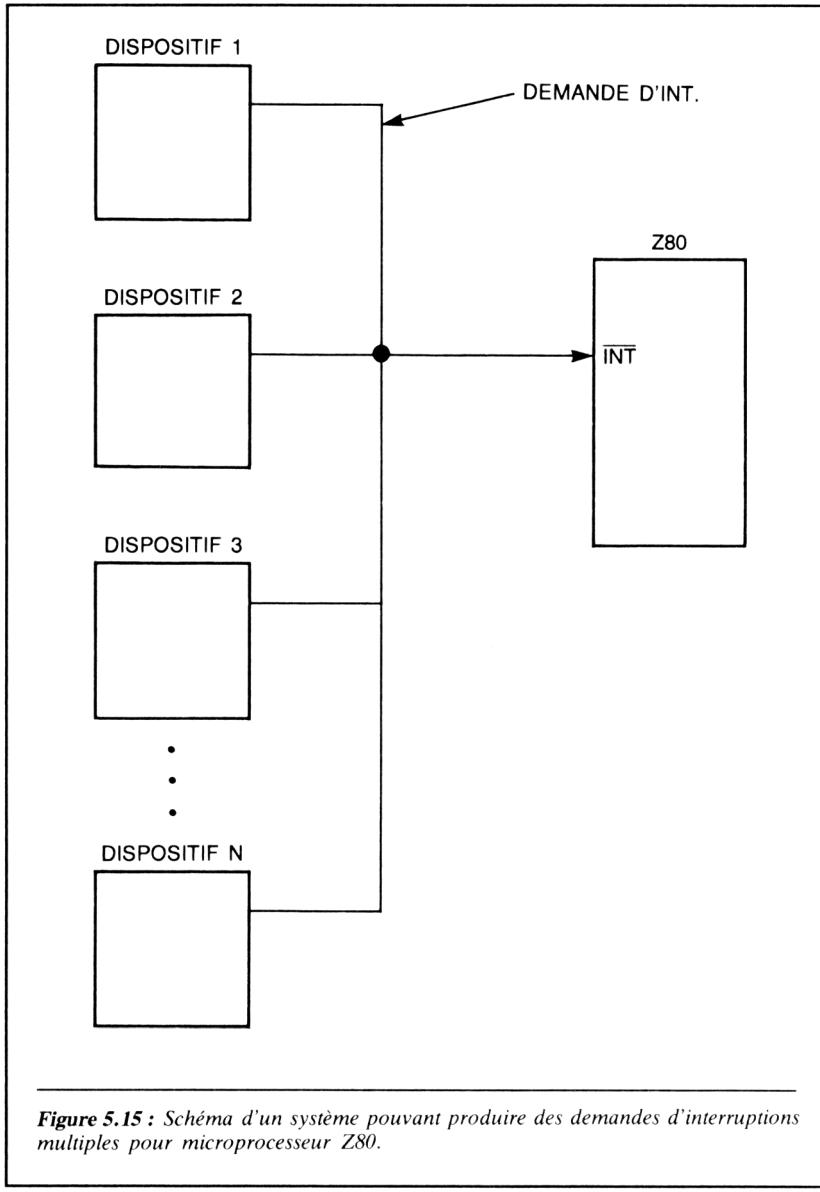


Figure 5.15 : *Schéma d'un système pouvant produire des demandes d'interruptions multiples pour microprocesseur Z80.*

schéma d'un système à interruptions multiples. Le CPU peut prendre ceci en compte de plusieurs façons. Nous allons analyser les principales techniques : scrutation, *(polling)* ; interruption prioritaire et chaînage, *(daisy chain).*

5.13 - Scrutation

La première technique que nous allons décrire est la *scrutation* (ou sondage). Supposons que le Z80 reçoive une demande d'interruption. La routine d'interruption ordonne au CPU de lire un certain octet, appelé *octet d'état*, dans chaque organe périphérique. Généralement, un bit de l'octet d'état va informer logiquement le Z80 si un organe particulier a demandé l'interruption. Si, cependant, plusieurs organes ont demandé l'interruption, le Z80 doit décider quel organe doit être servi d'abord. Ceci peut être effectué par scrutation.

Pendant la scrutation, le Z80 a sa broche d'interruption activée ce qui fait que tous les organes externes peuvent activer la ligne d'interruption. Cependant, il n'y a aucun moyen permettant au CPU de déterminer *directement* quel organe externe a activé la demande d'interruption. En fait tous les organes externes pourraient avoir leurs lignes d'interruptions connectées physiquement ensemble grâce à la structure en *drain ouvert* (dans le cas MOS) ou en *collecteur ouvert* (dans le cas TTL) des sorties. (Voir Figure 5.16.)

L'inconvénient de la technique de scrutation est que les interruptions ne sont pas gérées d'une façon rapide parce que le Z80 doit scruter chaque organe, c'est-à-dire lire un octet dans chacun d'eux. L'avantage de cette technique, cependant, est que les circuits sont simples et faciles à mettre en œuvre.

5.14 - Interruptions prioritaires

La technique d'interruption *prioritaire* permet au Z80 de gérer efficacement des demandes d'interruptions multiples. Voyons comment cela fonctionne.

Chaque organe extérieur est amené électriquement à mettre un octet particulier dans le bus de données lorsque son interruption a fait l'objet d'un accusé de réception par le Z80 (cette notion a été analysée en détail quand nous avons examiné les interruptions en mode 0 et en mode 2). Dans un système à interruptions multiples, chaque organe ne peut pas placer simplement l'octet de données sur le bus pendant un acquittement d'interruption parce que ceci entraînerait la présence de données erronées

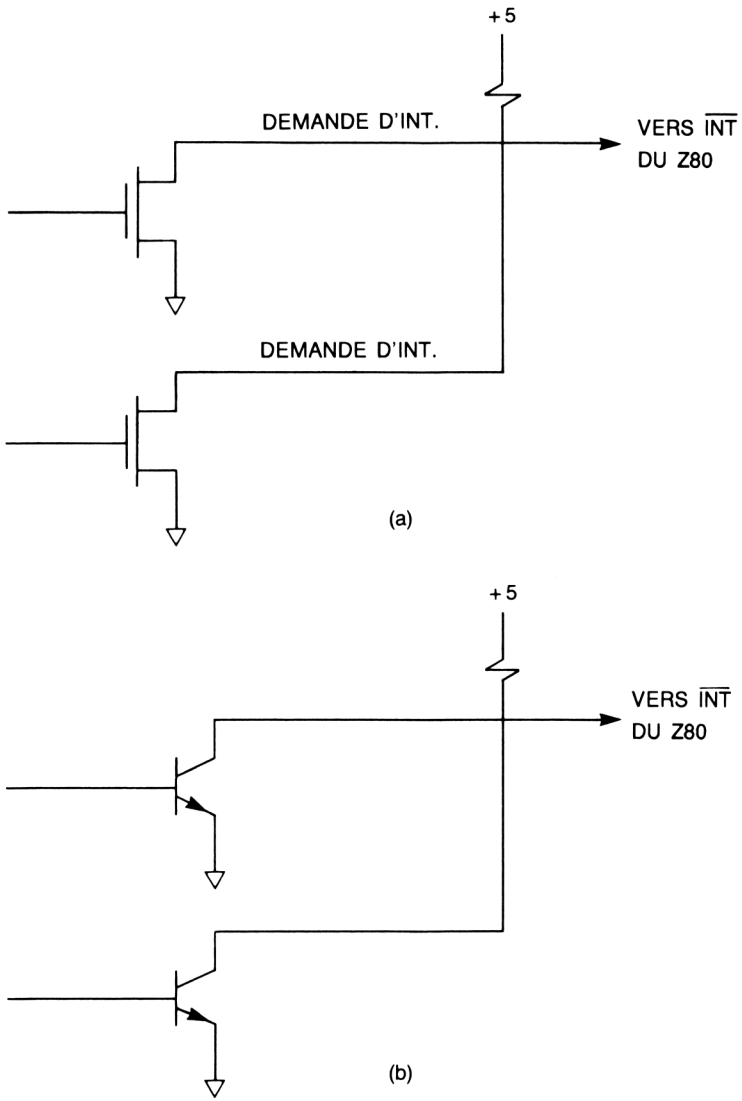


Figure 5.16 : *Schéma montrant la structure de sortie de dispositifs MOS et TTL. Ce type de structure de sortie est à drain ouvert ou à collecteur ouvert, ce qui signifie que ces sorties peuvent être connectées ensemble.*

dans le bus de données, plusieurs sources essayant de prendre le contrôle du bus. Il faut donc que le CPU accuse réception sélectivement aux demandes d'interruption.

Dans ce but, les circuits du système doivent établir un plan de priorité pour les organes externes. La logique de ce plan (montré dans le schéma de la Figure 5.17) est que chaque organe externe applique sa demande d'interruption à un bloc logique nommé *logique de priorité*. Ce bloc décide électriquement quelle ligne d'interruption externe doit être validée pour agir sur le CPU et *quand* elle doit être validée. Lorsque le CPU accuse réception de l'interruption, le bloc logique émet alors l'accusé de réception d'interruption vers l'organe externe qui a été autorisé à interrompre le CPU. En effet, la logique de priorité « dirige le trafic » en ce qui concerne les demandes d'interruptions et les accusés de réceptions.

L'avantage de ce type de plan d'interruption est que les interruptions sont traitées rapidement et efficacement par le CPU. L'inconvénient est qu'il faut des circuits supplémentaires pour réaliser ce plan.

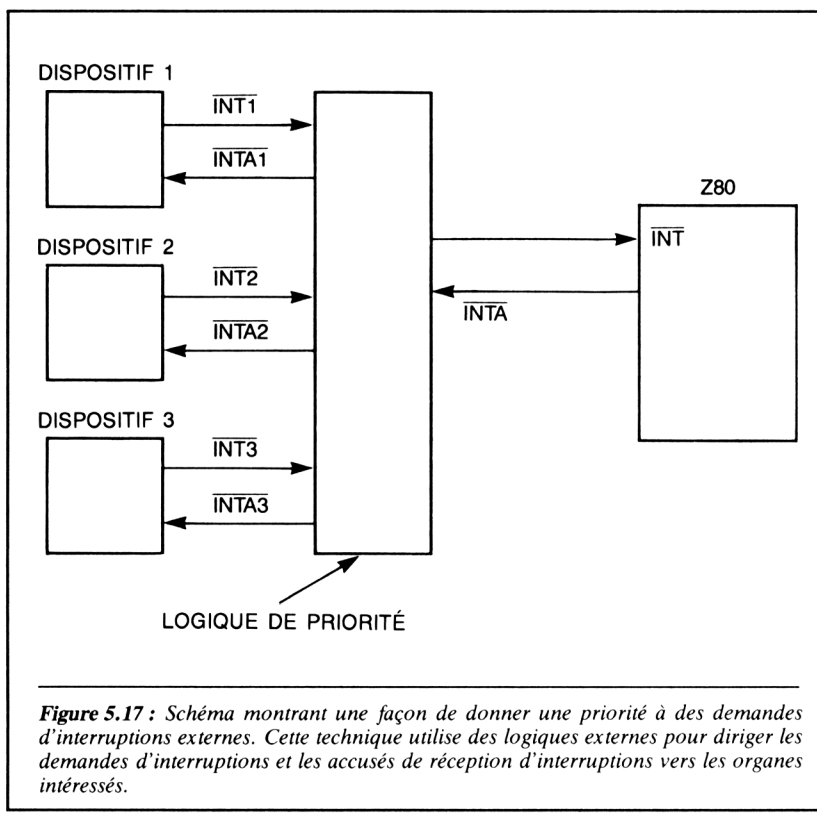


Figure 5.17 : *Schéma montrant une façon de donner une priorité à des demandes d'interruptions externes. Cette technique utilise des logiques externes pour diriger les demandes d'interruptions et les accusés de réception d'interruptions vers les organes intéressés.*

5.15 - Priorité à chaînage (*daisy chain*, configuration en pâquerette)

Une autre variante au plan d'interruption prioritaire est la structure en *chaînage*. Un schéma d'un système d'interruption à chaînage se trouve en Figure 5.18. Ceci interdit automatiquement aux organes externes d'adresser électriquement des interruptions d'un ordre plus bas dans la chaîne,

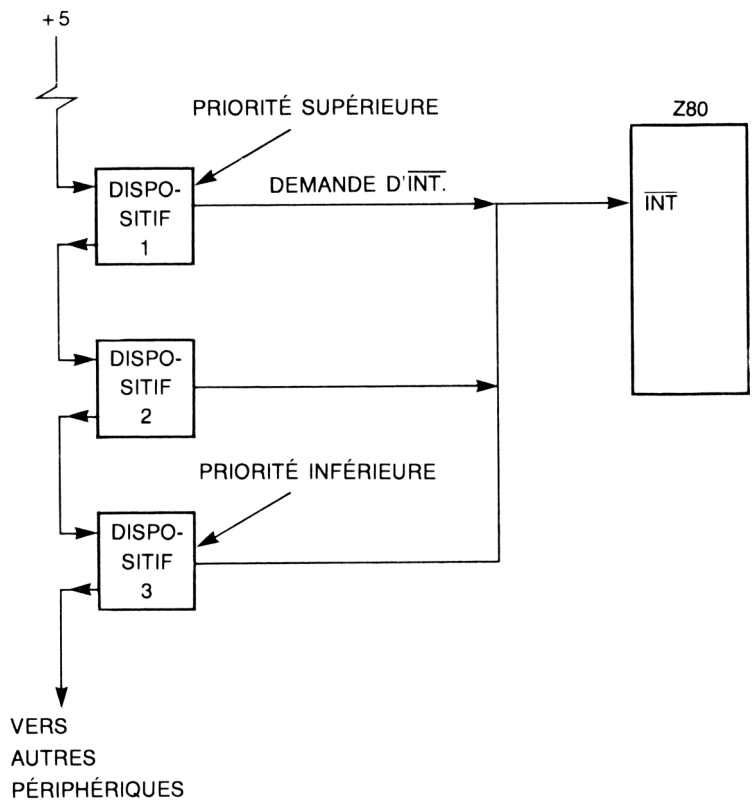


Figure 5.18 : *Schéma de la logique de priorité utilisée dans de nombreux systèmes à Z80. Cette technique emploie la structure en chaînage. Beaucoup de périphériques Z80 sont prévus pour être utilisés avec ce type de structure d'interruption.*

lorsqu'un organe d'un ordre plus haut effectue une demande d'interruption. (Voir Figure 5.18.) Si DEV 1 demande une interruption, DEV 2 et DEV 4 seront inhibés. Quand l'accusé de réception d'interruption est émis par le CPU, seul l'organe actif de la chaîne répond.

Si un organe de la chaîne est inhibé et qu'il existe les conditions qui causeraient une interruption si l'organe était validé, la demande sera conservée jusqu'à ce que l'organe soit autorisé — ainsi il n'y a pas de perte d'interruption dans la structure en chaînage.

Beaucoup d'organes périphériques du Z80 sont conçus pour être employés avec la solution du chaînage. Ces organes ont des circuits incorporés et des broches de dispositifs physiques qui rendent leur connexion en chaînage simple et facile. Dans le dernier chapitre, nous montrerons comment utiliser cette solution.

RESUME

Dans ce chapitre, nous avons examiné les différentes techniques d'interruption du Z80. Nous avons commencé en jetant un coup d'œil sur la notion d'interruption en général. Nous avons analysé ensuite chacun des trois modes d'interruption existant dans le Z80 : la scrutation, la priorité et le chaînage.

UTILISATION DU PIO 8255 AVEC LE Z80

CHAPITRE 6

INTRODUCTION

Le premier dispositif pour périphérique LSI (*Large Scale Integration*, intégration à grande échelle), que nous allons analyser maintenant, est l'adaptateur d'interface programmable 8255. Nous allons voir :

1. Comment il fonctionne.
2. Comment on le connecte électriquement aux bus du Z80.
3. Comment on peut le programmer pour fonctionner en tant que boîtier d'entrée-sortie à usages multiples.

6.1 - Aperçu général du 8255

Le 8255 est un boîtier à 40 broches, DIP (*Dual-In-line Package*, boîtier à double rangée de broches), LSI (*Large Scale Integration*, intégration à grande échelle), conçu pour effectuer de nombreuses fonctions d'interfaçage dans un système à microprocesseur. Contrairement à de nombreux dispositifs pour périphériques que nous allons analyser dans cet ouvrage, le 8255 n'a pas été conçu à l'origine pour être utilisé avec le microprocesseur Z80. Il a été mis en fabrication par Intel Corporation pour être employé avec le microprocesseur 8080.

Commençons par expliquer comment fonctionne le 8255 avant de le relier aux bus du Z80 et avant de le programmer. Une fois familiarisé avec le 8255 à ce niveau élémentaire, vous pourrez facilement explorer ses possibilités, notamment dans vos propres applications.

La Figure 6.1 donne le schéma du 8255. Examinons la fonction de chaque bloc de ce schéma.

Nous pouvons voir qu'il y a quatre blocs reliés aux lignes physiques du PIO qui peuvent être connectées à un dispositif externe. Ces lignes sont référencées PA0-PA7, PB0-PB7, PC0-PC7. Les groupes de signaux sont divisés logiquement en trois ports d'entrée-sortie différents, référencés port A (PA), port B (PB), port C (PC). Chaque port est associé à deux blocs

8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- **MCS-85™ Compatible 8255A-5**
- **24 Programmable I/O Pins**
- **Completely TTL Compatible**
- **Fully Compatible with Intel® Microprocessor Families**
- **Improved Timing Characteristics**
- **Direct Bit Set/Reset Capability Easing Control Application Interface**
- **40-Pin Dual In-Line Package**
- **Reduces System Package Count**
- **Improved DC Driving Capability**

The Intel® 8255A is a general purpose programmable I/O device designed for use with Intel® microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.

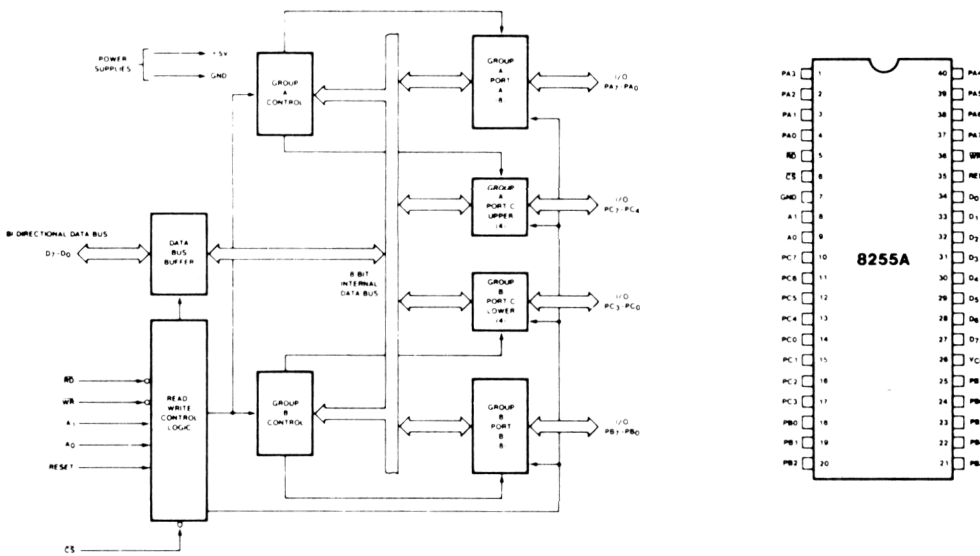


Figure 6.1 : Schéma et brochage du PIA 8255.

d'entrée-sortie séparés ; chaque bloc est connecté au bus de données interne du 8255. C'est par l'intermédiaire de ce bus interne que les informations sont échangées dans le 8255.

Dans la Figure 6.1, il y a deux blocs référencés commande du groupe A et commande du groupe B, qui déterminent le fonctionnement des trois ports d'entrée-sortie (le 8255 a plusieurs modes de fonctionnement différents ; ceux-ci doivent être déterminés par la programmation en écriture du CPU ou par des mots de commande). Remarquez que le groupe C du 8255 comporte deux ports de quatre bits. L'un d'eux est associé aux signaux du groupe A ; l'autre aux signaux du groupe B. (*Note :* Cette division en deux parties deviendra claire plus tard. Pour l'instant, contentons-nous de l'admettre.)

Les blocs logiques de la Figure 6.1 sont référencés tampon de bus de données et logique de commande lecture/écriture. Ces blocs fournissent l'interface électrique entre le microprocesseur Z80 et le 8255. Le tampon du bus de données sert aux lignes d'entrée et de sortie de données allant vers le bus de données du CPU et provenant de celui-ci. La logique de commande lecture/écriture envoie les données vers les registres internes appropriés, ou les reçoit de ceux-ci, avec le *timing* correct. Le trajet interne autorisé dépend du type de l'opération effectuée par le CPU ; autrement dit, cela dépend si l'opération est une lecture entrée-sortie ou une écriture entrée-sortie.

6.2 - Brochage du 8255

Nous allons énumérer les broches du 8255 et considérer la fonction de chacune d'elles. Ceci nous servira plus tard quand nous examinerons comment chaque broche se connecte aux bus du système à microprocesseur Z80. Ce brochage est donné en Figure 6.1. Examinons chaque broche :

D0-D7 Ce sont les lignes d'entrée et de sortie de données. Toutes les informations écrites ou lues passent par ces huit lignes de données.

$\overline{\text{CS}}$ (*Chip Select input,* entrée de sélection de boîtier) Chaque fois que cette entrée est au 0 logique, le microprocesseur peut lire et écrire électriquement dans le 8255.

$\overline{\text{RD}}$ (*ReaD input*, entrée de lecture) Chaque fois que cette ligne d'entrée est au 0 logique et que l'entrée $\overline{\text{CS}}$ est au 0 logique, les sorties de données du 8255 sont autorisées à aller dans le bus de données du système.

$\overline{\text{WR}}$ (*WRite input*, entrée d'écriture) Chaque fois que cette ligne d'entrée est au 0 logique et que l'entrée $\overline{\text{CS}}$ est au 0 logique, des données sont écrites dans le 8255 en provenance du bus de données du système.

A0-A1 (*Entrées d'adresses*) La combinaison logique de ces deux lignes détermine dans quel registre interne du 8255 se font les écriture ou lecture de données.

$\overline{\text{RESET}}$ Chaque fois que cette ligne d'entrée est au 1 logique, le 8255 est mis en état de réinitialisation. Tous les ports périphériques sont mis en entrées.

PA0-PA7 Ces lignes de signaux servent de port d'entrée-sortie à 8 bits, pouvant être relié à des périphériques.

PB0-PB7 Ces lignes de signaux servent de port d'entrée-sortie à 8 bits, pouvant être relié à des périphériques.

PC0-PC7 Ces lignes de signaux servent de port d'entrée-sortie à 8 bits, pouvant être relié à des périphériques. De plus, ce groupe de signaux peut être divisé en deux groupes de 4 bits servant à commander les lignes PB0-PB7 et PA0-PA7 dans certains modes de fonctionnement du 8255.

Maintenant que nous sommes familiarisés avec les broches du 8255, utilisons ces renseignements pour le connecter aux bus du système à Z80.

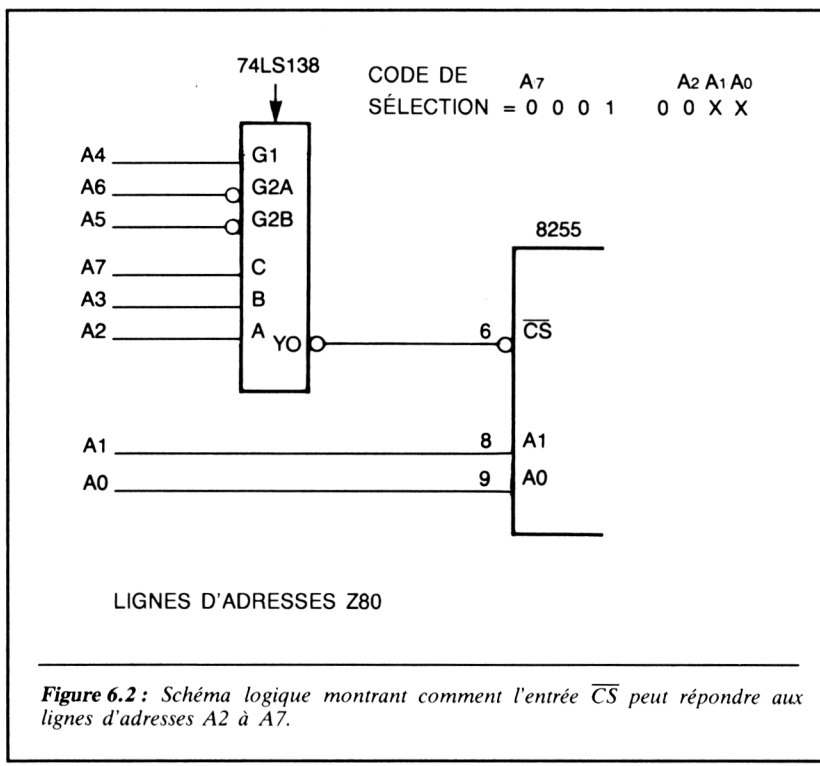


Figure 6.2 : *Schéma logique montrant comment l'entrée $\overline{CS}$ peut répondre aux lignes d'adresses A2 à A7.*

6.3 - Liaison du 8255 avec le CPU Z80

Utilisons maintenant le 8255 en port de sortie dans le système à Z80. Nous allons le connecter au CPU Z80 et nous supposerons que toutes les communications se présentent comme des opérations d'entrée-sortie. Le 8255 a deux entrées d'adresses, référencées A0 et A1 ; ceci implique que le 8255 contient quatre adresses d'entrée-sortie distinctes. En utilisation normale, les entrées A0 et A1 du boîtier sont connectées aux lignes de sorties d'adresses mémoire A0 et A1 du CPU Z80.

La ligne d'entrée $\overline{CS}$ du 8255 sert à placer logiquement le boîtier dans l'espace d'entrée-sortie du système. Supposons, par exemple, que l'on attribue les adresses d'entrée-sortie égales à 10H, 11H, 12H, 13H. On peut réaliser cet adressage logiquement et électriquement en employant un circuit comme celui de la Figure 6.2.

Nous avons vu au Chapitre 3 qu'une opération d'entrée-sortie nécessite un décodage de 8 lignes d'adresses de poids faible du Z80, les lignes A7-A0. Dans la Figure 6.2, l'entrée $\overline{CS}$ ne devient active que lorsque les 6 bits d'adresses de poids fort (A7-A2) sont égaux à 000100XX (les deux bits d'adresses de poids faible servent à déterminer lequel des quatre registres

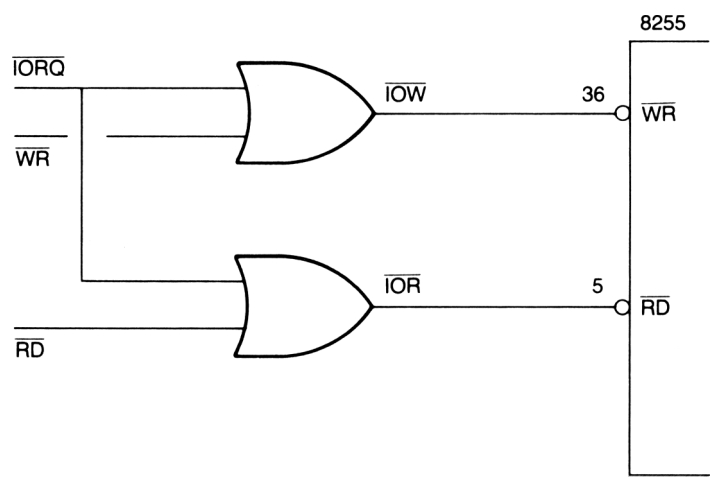


Figure 6.3 : *Les entrées $\overline{WR}$ et $\overline{RD}$ du 8255 peuvent être connectées physiquement aux lignes de commandes $\overline{IOR}$ et $\overline{IOW}$ du système. Ces lignes du système deviennent actives chaque fois que le Z80 communique avec un dispositif d'entrée-sortie.*

internes est employé pour écrire ou lire). Cette structure d'entrée-sortie est quelquefois nommée *port d'entrée-sortie du dispositif.* Autrement dit, l'adresse du dispositif comporte quatre adresses de port.

L'étape suivante consiste à connecter les lignes d'entrées $\overline{RD}$ et $\overline{WR}$ aux signaux de commande $\overline{IOR}$ et $\overline{IOW}$ du système à Z80. Il faut le faire car,

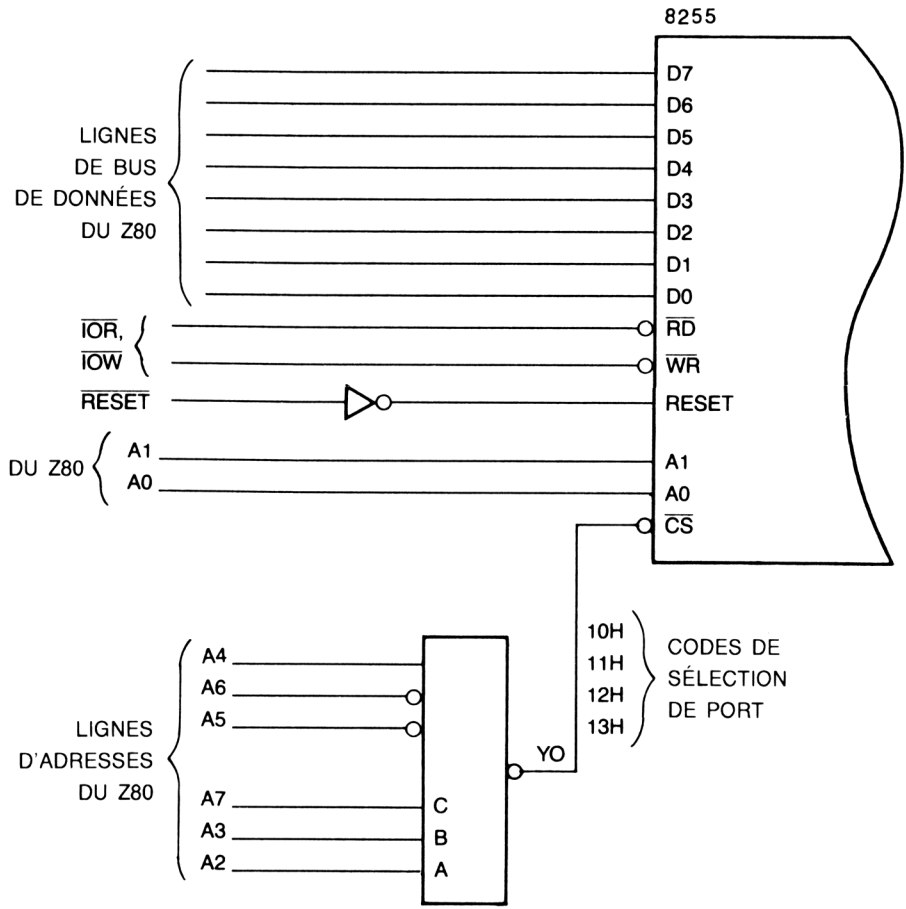


Figure 6.4 : *Schéma complet montrant les interconnexions entre Z80 et 8255.*

dans notre application, les entrées $\overline{RD}$ et $\overline{WR}$ ne peuvent pas être connectées directement aux sorties $\overline{RD}$ et $\overline{WR}$ du Z80. Ceci est dû au fait que, dans le Z80, ces lignes sont actives aussi pendant les opérations en mémoire et si votre application n'utilise pas d'entrée-sortie, cette connexion ne sera pas valide. La Figure 6.3 montre une façon de connecter $\overline{IOR}$ et $\overline{IOW}$ au 8255.

Le dernier signal de commande à connecter au 8255 est l'entrée *reset* (remise à zéro). Un point important à noter au sujet de l'entrée de remise à zéro est qu'elle est active au 1 logique. L'entrée de remise à zéro du Z80 est active au 0 logique. Si, donc, on doit l'employer pour le 8255, il est nécessaire de l'inverser après l'avoir appliquée au Z80.

Les lignes de données D0-D7 peuvent être connectées directement aux lignes d'entrée de données du 8255. La Figure 6.4 montre toutes les liaisons entre Z80 et 8255. On suppose ici que cette application n'exige pas de tampon de données. Autrement dit, la charge du bus de données du système peut être supportée directement par les lignes de sortie de données du 8255.

6.4 - Registres de lecture et d'écriture du 8255

Maintenant que nous avons connecté le 8255 au Z80, voyons comment programmer le dispositif pour qu'il fonctionne dans notre application. Nous allons commencer par examiner les quatre registres internes (c'est-à-dire les quatre registres de lecture et d'écriture) du 8255. Dans notre exemple de décodage, l'adresse de ces registres est 10H, 11H, 12H et 13H. Les définitions fondamentales des registres sont :

BROCHES				NOM DU REGISTRE
$\overline{RD}$	$\overline{WR}$	A1	A0	
1	0	0	0	écrire données dans PORT A
0	1	0	0	lire données dans PORT A
1	0	0	1	écrire données dans PORT B
0	1	0	1	lire données dans PORT B
1	0	1	0	écrire données dans PORT C
0	1	1	0	lire données dans PORT C
1	0	1	1	écrire mot de commande
0	1	1	1	illégal

La fonction des registres 0-2 est définie par le mot écrit dans le registre de commande 3. La Figure 6.5 montre la définition des bits du registre de commande. Examinons maintenant quelques-uns des principaux modes de fonctionnement du 8255 en considérant plusieurs exemples de programma-

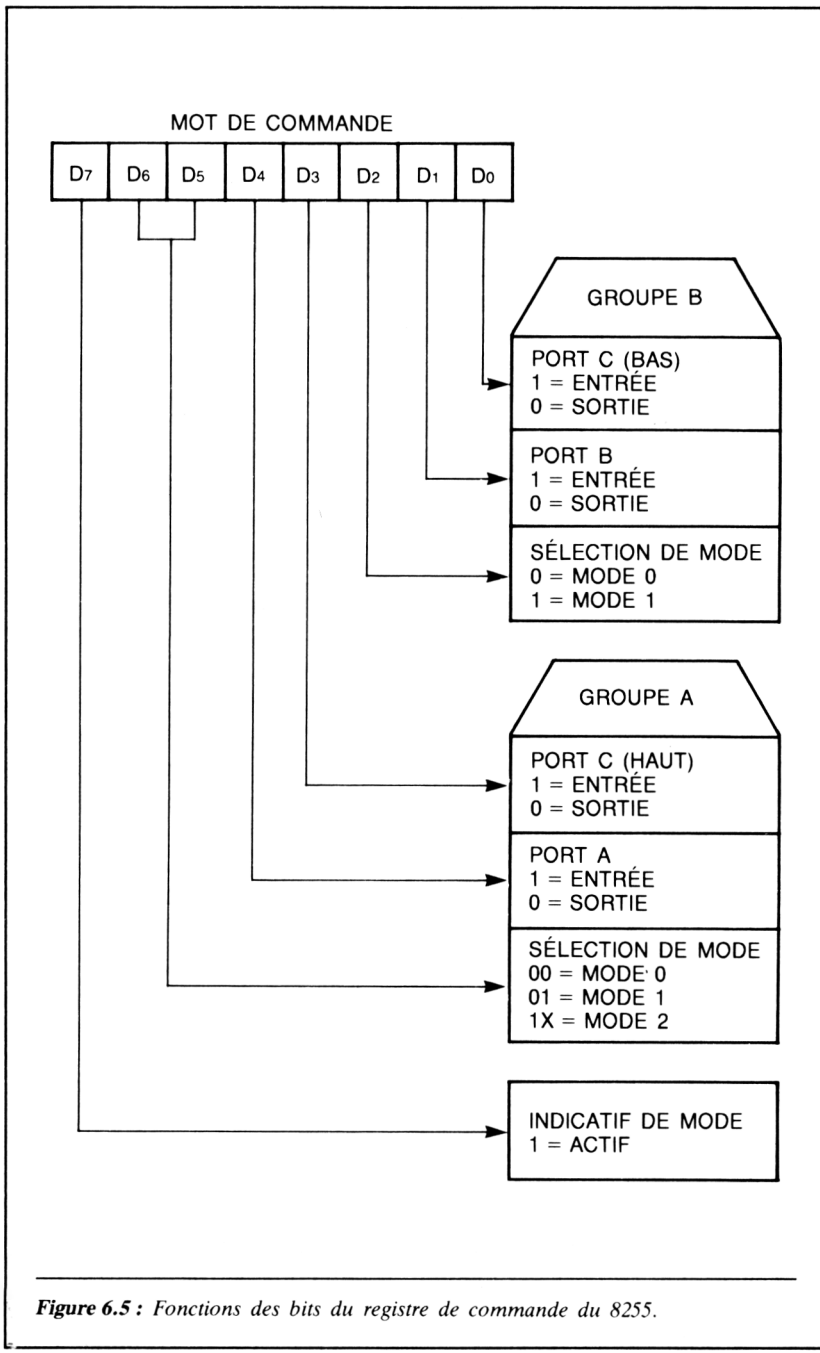


Figure 6.5 : *Fonctions des bits du registre de commande du 8255.*

tion de Z80 pour le dispositif. Nous allons maintenant analyser les trois modes de fonctionnement du 8255 et voir comment on utilise celui-ci dans chaque cas.

6.5 - Mode 0 - Registre d'entrée-sortie fondamental

Pour établir le mode 0 pour registre d'entrée-sortie fondamental, le programmeur doit d'abord écrire un mot de commande dans le registre de commande. Ce mot va déterminer comment les registres doivent être employés dans le 8255. Les bits du registre de commande, qui servent à programmer la fonction standard d'entrée-sortie, doivent apparaître comme ceci :

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0

En nous reportant à la Figure 6.5, nous pouvons voir que :

- Le bit D7 définit ce mot comme mot de commande.
- Les bits D6 et D5 définissent le mode de fonctionnement du port A du 8255. C'est le mode 0.
- Le bit D4 = 0 indique que le port A est en sortie.
- Le bit D3 = 0 met les 4 bits de poids fort du port C en sorties.
- Le bit D2 établit le mode pour le port B. Un 0 logique met le port B en port de sortie.
- Le bit D1 = 0 met le port B en port de sortie.
- Le bit D0 = 0 met les 4 bits de poids faible du port C en port de sortie.

Ce mot de commande écrit dans le 8255 détermine les trois ports, A, B et C, comme ports de sortie. Ceci donne au 8255 vingt-quatre lignes de sortie distinctes qui peuvent servir à interfacer les dispositifs externes. Les instructions du Z80 pour configurer le 8255 en mode 0 sont :

```
LD A,80H            ETABLIR LE MOT DE COMMANDE
OUT (13H),A         ENVOYER LE MOT DE COMMANDE AU 8255
```

Une fois le 8255 programmé avec ces deux instructions du Z80, on peut écrire des données dans n'importe quel port de sortie en employant des instructions OUT du Z80. Supposons, par exemple, que nous voulions écrire 23H dans les sorties du port A, 41H dans le port B et 73H dans les sorties du port C. Pour accomplir cela, nous utiliserons une séquence d'instructions telle que :

```
LD A,23H            ETABLIR DONNEES PORT A
OUT (10H),A         SORTIR DONNEES VERS 8255
LD A,41H            ETABLIR DONNEES PORT B
OUT (11H),A         SORTIR DONNEES VERS 8255
LD A,73H            ETABLIR DONNEES PORT C
OUT (12H),A         SORTIR DONNEES VERS 8255
```

Après exécution de toutes ces instructions, les ports de sortie A, B et C du 8255 seront programmés aux valeurs de données spécifiées. C'est une manière commode de mettre trois ports de sortie distincts dans un seul boîtier.

On peut programmer les ports en combinaison d'entrées et de sorties. Par exemple, les ports A et C peuvent être programmés en ports de sortie et le port B en port d'entrée. En nous reportant à la Figure 6.5, le mot de commande établissant le 8255 dans cette configuration serait :

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	0

Après programmation de ce mot de commande dans le registre 3 du 8255, le boîtier sera configuré logiquement comme le montre la Figure 6.6. On peut employer une instruction IN pour lire les entrée de données dans le port B :

```
IN A,(11H)          LIRE DONNEES DANS PORT B
```

Après que le 8255 a reçu la configuration correcte, il devient facile de lire ou d'écrire des données dans n'importe quel port du boîtier. Il y a plusieurs combinaisons différentes pour configurer le 8255 en entrée-sortie en mode 0. La Figure 6.7 montre toutes ces combinaisons, ainsi que le mot de commande approprié à la programmation du 8255.

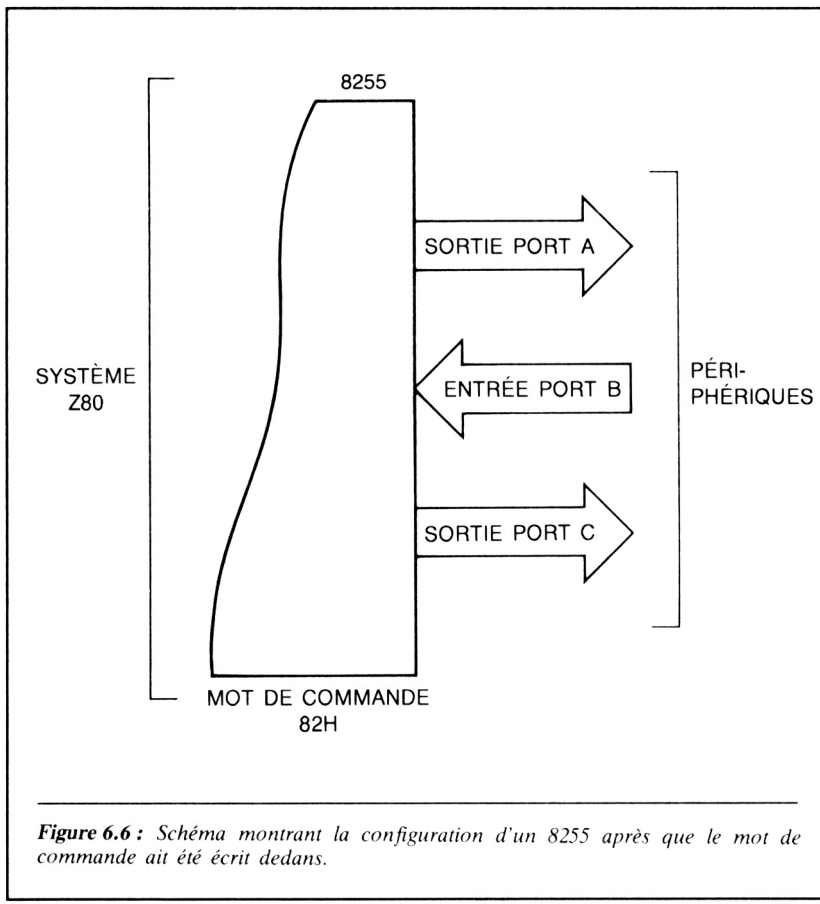


Figure 6.6 : *Schéma montrant la configuration d'un 8255 après que le mot de commande ait été écrit dedans.*

6.6 - Exemple de mode 0

Comme exemple d'utilisation du Z80 avec le 8255 en mode 0, examinons maintenant une simple interface de clavier. Nous allons examiner à la fois les circuits et le logiciel du clavier. Ce dernier est organisé en matrice 4 × 4 de contacts SPST (*Single Pole, Single Throw,* unipolaire à une direction). Nous emploierons le 8255 pour interfacer les contacts du clavier avec le Z80. Le schéma est donné en Figure 6.8. Voici comment fonctionnent circuit et programme.

Les entrées du port B sont reliées aux colonnes du clavier. Les entrées du port A sont reliées aux rangées du clavier. Toutes les entrées de rangées sont rappelées au + 5 v par l'intermédiaire de résistances de 10 kilohms.

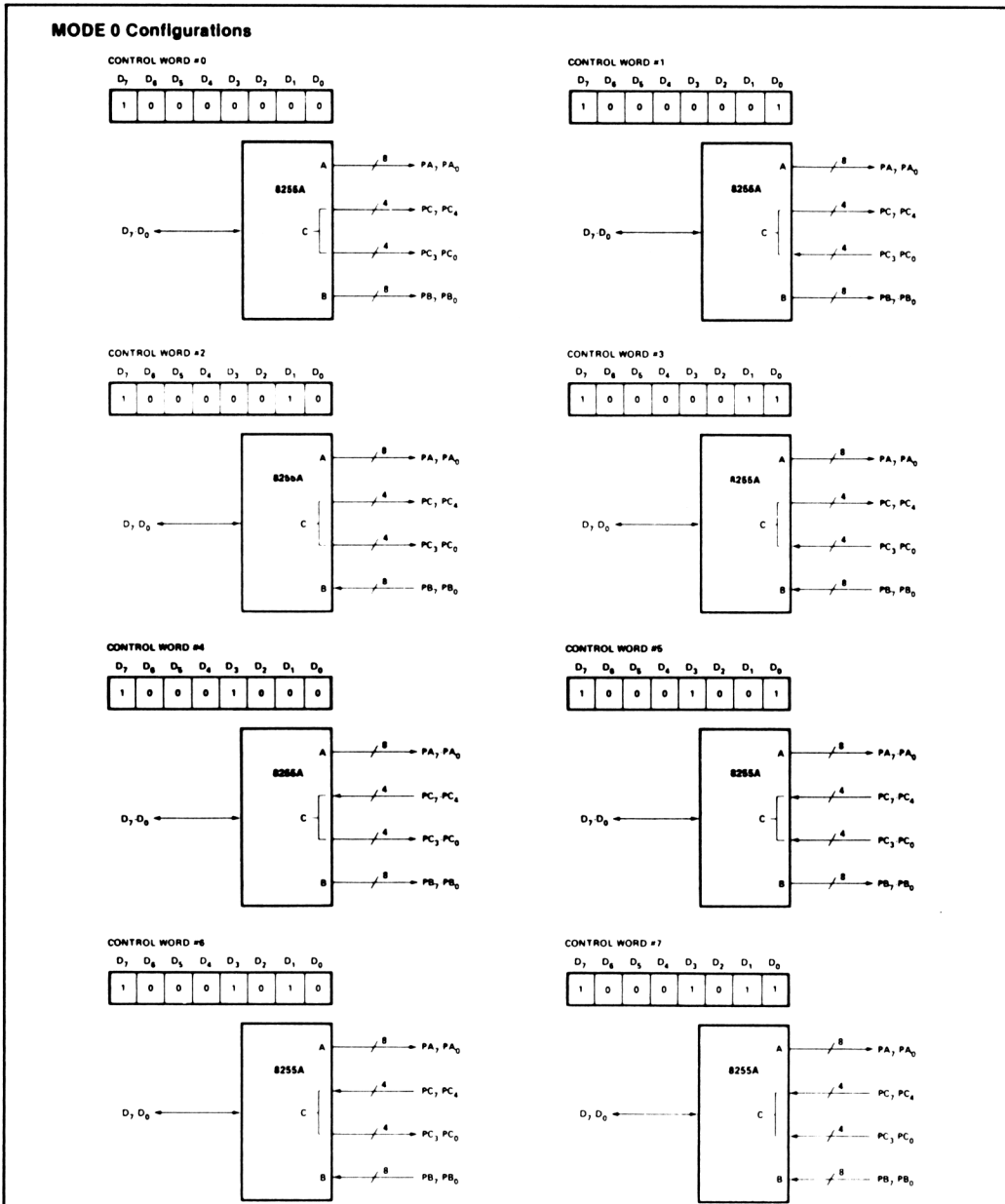


Figure 6.7 : Voici toutes les configurations possibles utilisables en mode 0.

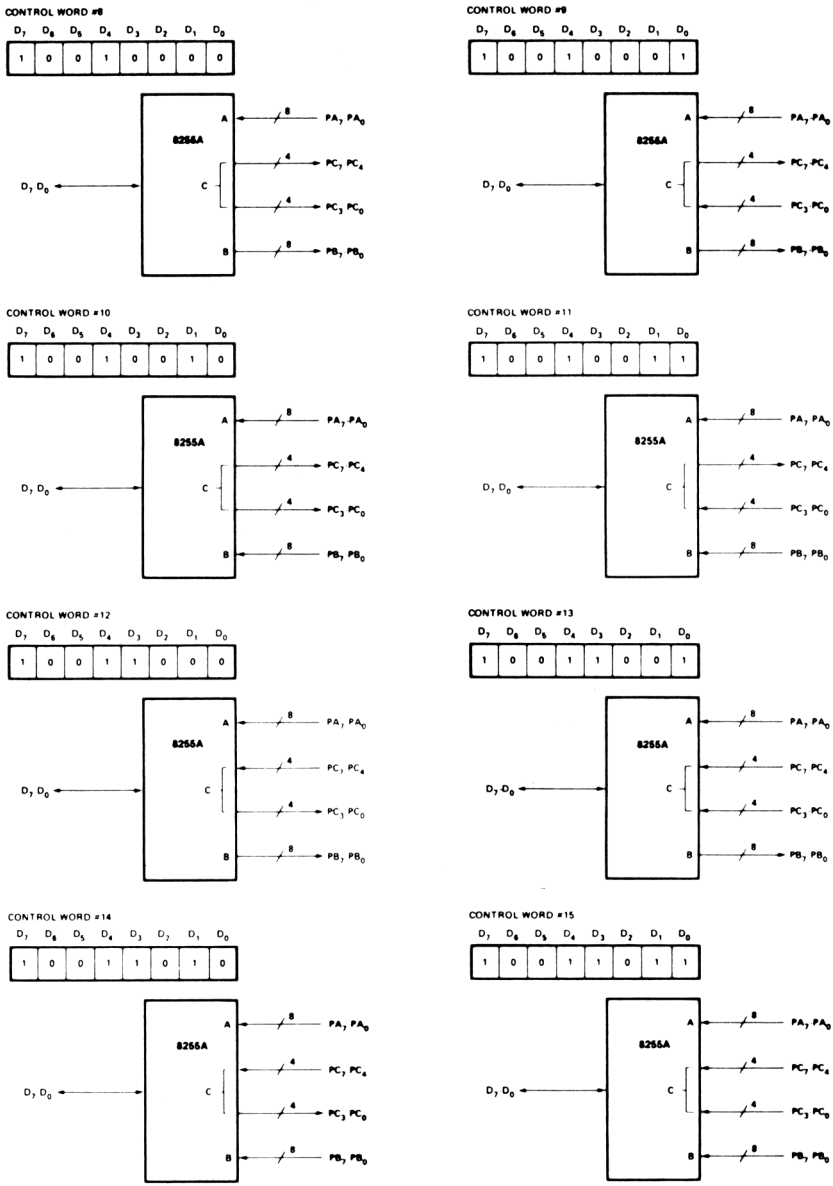


Figure 6.7 : *Configurations possibles utilisables en mode 0 (suite).*

Les colonnes sont forcées au 0 logique l'une après l'autre. Après que chaque colonne ait été mise au 0 logique, le port d'entrée A est lu. Si une lecture de bit du port d'entrée est un 0 logique, c'est qu'une touche a été enfoncée. On peut déterminer cette touche en notant le bit qui est au 0 logique et la colonne qui est active. La Figure 6.9 montre un programme qui fonctionne de cette façon.

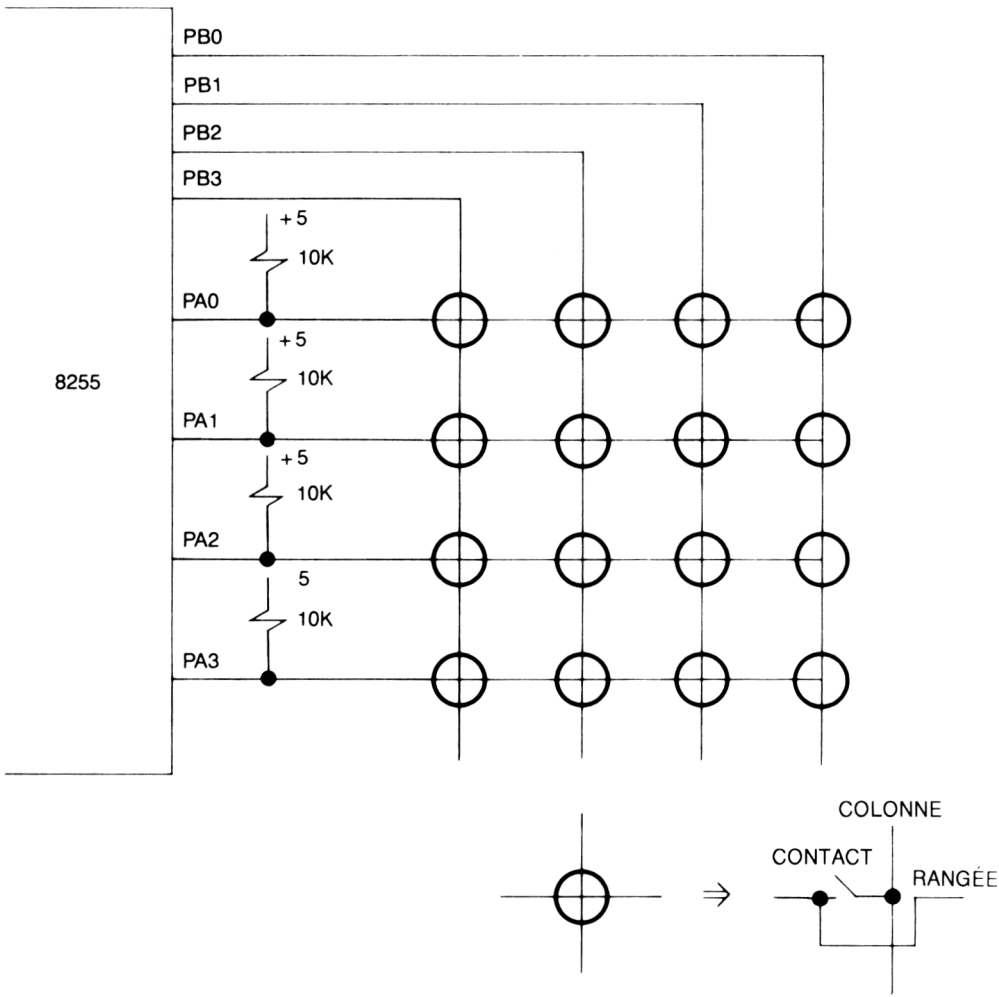


Figure 6.8 : *Schéma montrant les interconnexions d'une matrice de clavier 4 × 4 simple interfacée à un 8255.*

```
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;     PROGRAMME POUR DETECTER LA MANŒUVRE D'UNE TOUCHE
;     DANS LE CIRCUIT DE LA FIGURE 6.8
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   LE PORT B EST LE PORT DE SORTIE
;   LES PORTS A ET C SONT LES PORTS D'ENTREE
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
0010              PORTA EQU  10H
0011              PORTB EQU  11H
0012              PORTC EQU  12H
0013              CONP  EQU  13H
0099              CONWD EQU  99H
;
1800                    CODE 1800H
;
1800   3E99             LD A,CONWD
1802   D313             OUT (CONP),A   ;ETABLIR LE 8255
;
1804   0E11             LD C,PORTB     ;CREG = PORTB ADD
;
1806   06FE       COL   LD B,0FEH      ;COLONNE 0 ACTIVE
1808   ED41       COL1  OUT (C),B      ;ACTIVER COLONNE
180A   DB10             IN A,(PORTA)   ;LIRE LES RANGEES
180C   2F               CPL            ;COMPLEMENTER MOT D'ENTREE
180D   E60F             AND 0FH        ;MASQUER BITS INUTILISES
180F   FE00             CP 00H         ;UN BIT = 12 ?
1811   C21F18           JP NZ,KEYIN    ;SI OUI, TOUCHE APPUYEE
1814   CB00             RLC B          ;DECALER B A GAUCHE, COLONNE SUIVANTE ACTIVE
1816   78               LD A,B
1817   FEEF             CP 0EFH        ;VERIFIER DERNIERE COLONNE ACTIVE
1819   C20818           JP NZ,COL1     ;PAS LA DERNIERE, ACTIVER COLONNE SUIVANTE
181C   C30618           JP COL         ;DERNIERE, ACTIVER PREMIERE COLONNE
;
;   MANIPULER TOUCHE APPUYEE, NECESSITE COLONNE ET RANGEE
;

181F   4F         KEYIN LD C,A         ;POS. DE RANGEE = 1 DANS REG. C
1820   78               LD A,H         ;POS. DE COLONNE DANS REG. A
1821   2F               CPL            ;INVERSER OCTET DE POS.
1822   E60F             AND 0FH        ;BITS INUTILISES = 0
1824   C9               RET
;
;   RETOUR DE ROUTINE AVEC
;   REG. A = POS. DE COLONNE, 1 DANS BIT DE COLONNE ACTIVE
;   REG. C = POS. DE RANGEE, 1 DANS BIT DE RANGEE ACTIVE
;
;
                         FIN
```

Figure 6.9 : *Programme de Z80 pour explorer le clavier de la Figure 6.8 et déterminer la touche qui a été appuyée.*

Bien que cet exemple ne concerne qu'une matrice simple 4 × 4, on peut l'étendre facilement à une matrice de n'importe quelle dimension. L'emploi du 8255 est une manière simple d'avoir l'interface nécessaire.

6.7 - Fonctionnement en mode 1 du 8255

Un autre mode de fonctionnement du 8255 est celui d'un dispositif d'entrée-sortie utilisé dans un contexte de *handshaking*. Les ports A et B sont les ports de données de 8 bits ; les bits de poids fort du port C servent de lignes de *handshaking* pour le port A, les bits de poids faible du port C servent de lignes de *handshaking* pour le port B.

L'idée principale d'un transfert d'entrée-sortie utilisant un *handshaking* est la suivante : le périphérique informe électriquement le 8255 que les données sont prêtes, soit à être envoyées, soit à être prises. Les lignes de *handshaking* servent à fournir cette information. (Voir Figure 6.10.)

Dans le schéma (a) de la Figure 6.10, les données sont envoyées au 8255 par le périphérique. Avant que le périphérique n'écrive des données dans le 8255, il doit aussi vérifier l'indicateur de remplissage du tampon d'entrée. Si cet indicateur est « vrai », les données du tampon du 8255 n'ont pas été lues par le Z80. Cela signifie que les données ont été envoyées au 8255 par le dispositif extérieur, mais que le Z80 ne les a pas lues. Si l'indicateur est « faux », le dispositif extérieur va écrire les données dans le 8255 et l'indicateur de remplissage du tampon d'entrée va devenir « vrai ». Quand le Z80 va lire les données, l'indicateur de remplissage du tampon d'entrée va devenir « faux ».

Dans le schéma (b) de la Figure 6.10, le 8255 va bientôt envoyer des données vers le périphérique. Cependant, avant de pouvoir envoyer celles-ci, le 8255 doit d'abord vérifier si l'indicateur de remplissage du tampon de sortie est établi. Ce drapeau signale au dispositif extérieur que le 8255 a des données prêtes à être saisies par le dispositif extérieur. Quand celui-ci prend en compte les données du 8255, l'indicateur de remplissage du tampon de sortie devient « faux » signalant ainsi que le dispositif extérieur a saisi les données. A ce moment-là, le Z80 peut envoyer d'autres données au tampon de sortie, pour qu'elles soient saisies par le dispositif extérieur.

Cette procédure de *handshaking* pour manipuler les données d'un dispositif à un autre est très utile dans les applications où le dispositif extérieur est plus lent que le microprocesseur du système. Avec cette technique, le microprocesseur peut mettre des données dans le tampon de sortie, puis effectuer d'autres tâches. Quand le dispositif extérieur lent a saisi les données, le microprocesseur peut en envoyer d'autres. Examinons en détail comment le 8255 fonctionne en *handshaking*.

Supposons que le 8255 envoie des données au dispositif extérieur par le port A et qu'il reçoive des données par le port B. Ceci pourrait être inversé,

c'est-à-dire que le port A pourrait recevoir des données et que le port B pourrait en envoyer. Une autre solution consisterait à ce que les ports envoient tous les deux des données ou bien en reçoivent tous les deux. L'idée essentielle est que les ports A et B peuvent être utilisés tous les deux en entrée ou en sortie, avec procédure de *handshaking,* pour entrée ou sortie des données. En outre, chaque port est indépendant de l'autre. Il suffit de modifier le mot de commande pour établir le 8255 dans le fonctionnement de registre approprié.

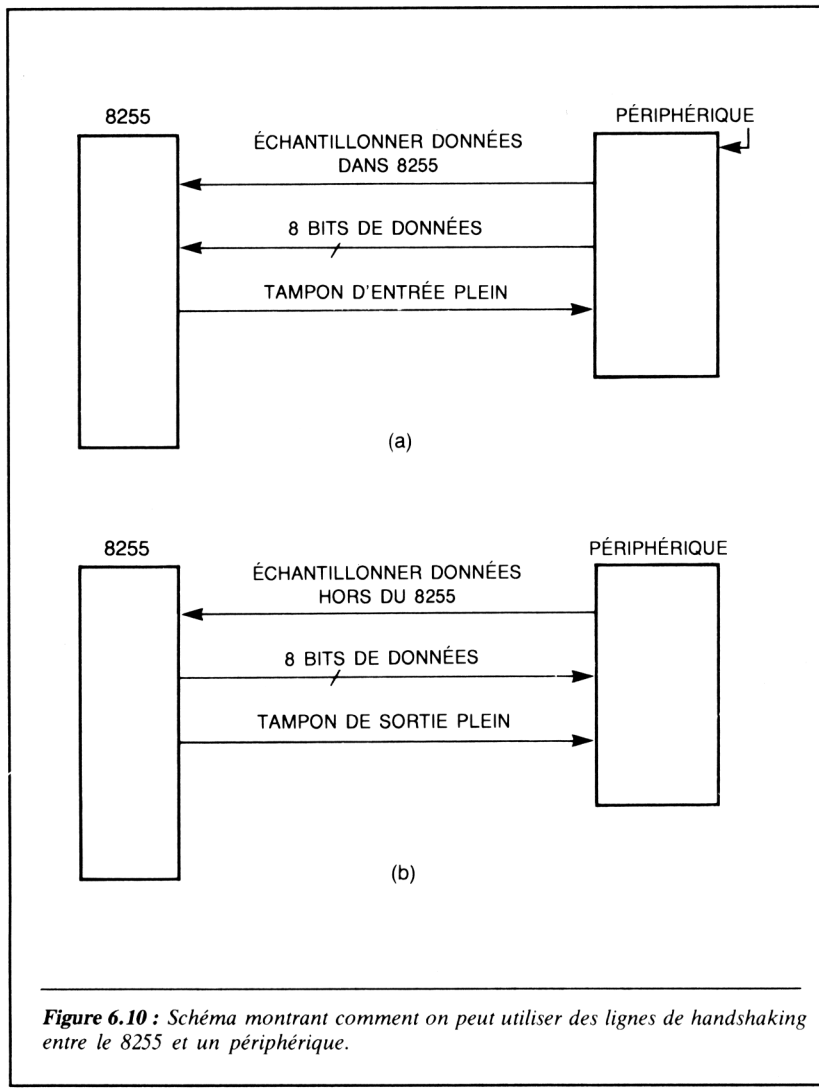


Figure 6.10 : *Schéma montrant comment on peut utiliser des lignes de handshaking entre le 8255 et un périphérique.*

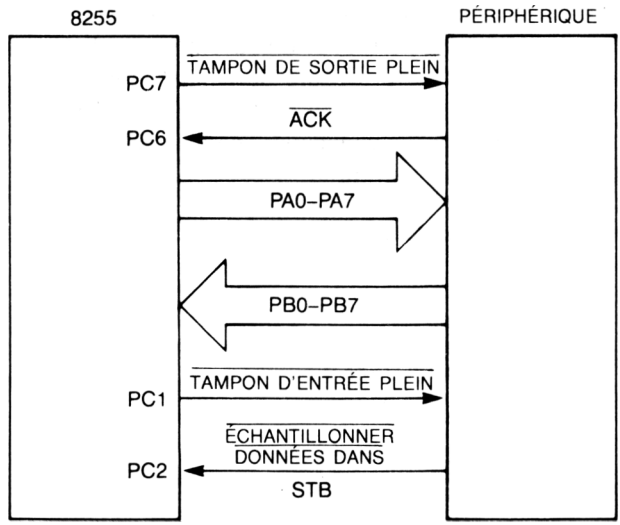


Figure 6.11 : *Configuration de 8255 pour envoyer des données des lignes du port A vers le périphérique et entrer des données du périphérique dans les lignes du port B.*

	OUT	IN
PC0	$INTR_B$	$INTR_B$
PC1	IBF_B	$\overline{OBF}_B$
PC2	$\overline{STB}_B$	$\overline{ACK}_B$
PC3	$INTR_A$	$INTR_A$
PC4	$\overline{STB}_A$	I/O
PC5	IBF_A	I/O
PC6	I/O	$\overline{ACK}_A$
PC7	I/O	$\overline{OBF}_A$

Figure 6.12 : *Définition des broches du port C en entrée et en sortie, en mode 1.*

Le schéma de la Figure 6.11 montre la configuration que nous désirons pour cette application. Pour mettre le 8255 dans cette configuration, le mot de commande convenant à ce mode doit être écrit dans le dispositif avant son utilisation. Le mot de commande pour cette configuration serait 10100110 ou 0A3H.

Dans cette figure, les sorties de données sont sur les broches PA0 à PA7 du 8255. L'indicateur de remplissage du tampon de sortie est en PC7 ; l'accusé de réception du dispositif extérieur est en PC6 ; les entrées de données sont en PB0 à PB7 ; l'indicateur de remplissage du tampon d'entrée est en PC1 ; et le signal de mise en compte des données dans le 8255 est en PC2. La Figure 6.12 donne les définitions des broches du port C pour entrée et sortie en mode 1.

Maintenant que nous avons défini les circuits, examinons le logiciel qui permet au 8255 de fonctionner. Nous supposerons ici qu'il n'y aura pas d'interruptions. Le Z80 va scruter les lignes d'état du 8255 pour vérifier si des données ont été reçues dans le port d'entrée ou prises dans le port de sortie. Nous supposerons, en outre, que le dispositif extérieur est capable, électriquement, de prendre les données et de les envoyer au 8255.

Il est important de remarquer que le 8255 ne fournit que la moitié de la solution. L'autre moitié est que le dispositif extérieur doit être mis en communication électrique d'une manière compatible avec le 8255. Les Figures 6.13 (a) et 6.13 (b) montrent le *timing* d'un transfert type avec le 8255 dans la configuration requise.

Pour envoyer un caractère au dispositif extérieur, le 8255 doit examiner d'abord la ligne de l'indicateur de remplissage du tampon de sortie, PC7, pour dire si elle est au 1 logique. On le fait en lisant les données du port C, en employant une instruction d'entrée. Si le port C, bit D7, est au 1 logique, le dispositif extérieur a alors pris les données qui étaient présentes dans le tampon de sortie. Si le bit D7 du port C est au 0 logique, le Z80 ne doit pas envoyer d'autres données. Quand le bit D7 du port C va au 1 logique, le Z80 peut à nouveau écrire des données dans le registre de données du port A. A ce moment-là, la ligne du tampon de sortie du port C, bit D6, va au 0 logique. Ceci indique au dispositif extérieur que les données sont présentes et peuvent être prises. Le dispositif extérieur répond à la condition « vrai » de cette ligne en prenant en compte la ligne d'entrée ACK qui est le bit D7 du port C indiqué précédemment. La Figure 6.14 donne l'ordinogramme du fonctionnement en sortie complète, en mode *handshaking*. Les mnémoniques du Z80, permettant de réaliser cet ordinogramme, apparaissent en Figure 6.15.

Voyons maintenant comment le Z80 lit les données provenant du dispositif extérieur par l'intermédiaire du 8255. La première opération qui se déroule est la vérification par le Z80 de la ligne de l'indicateur de remplissage du tampon d'entrée, bit D1 du port C. Quand cette ligne est au

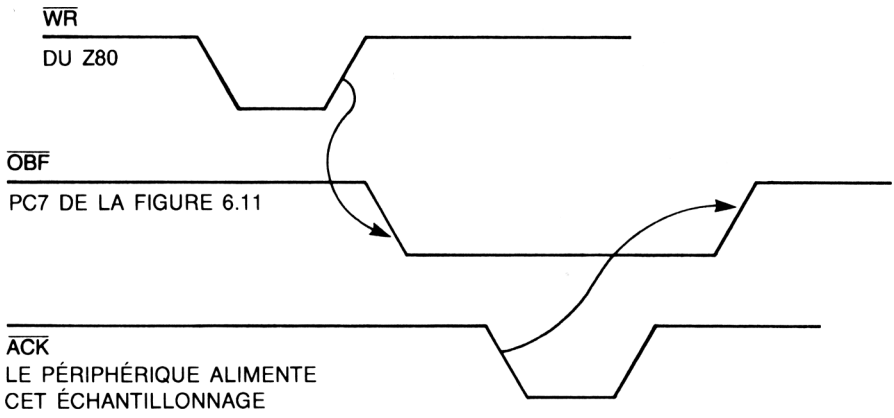


Figure 6.13 (a) : *Diagramme des temps d'un transfert type dans les lignes du port A du 8255, dans la configuration de la Figure 6.12.*

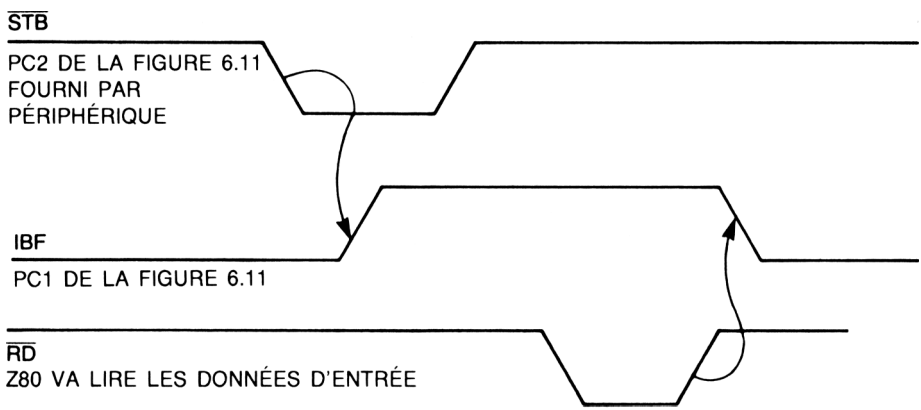


Figure 6.13 (b) : *Diagramme des temps d'un transfert type dans les lignes du port B du 8255, dans la configuration de la Figure 6.12.*

1 logique, ceci indique que le dispositif extérieur a pris en compte les données du 8255 par la ligne d'entrée $\overline{STB}$, bit D2 du port C. A ce moment-là, le Z80 lit le registre de données du port B, en employant l'instruction IN. La ligne du tampon d'entrée, bit D1 du port C, va alors au 0 logique, indiquant ainsi au dispositif extérieur qu'un autre octet peut être pris en

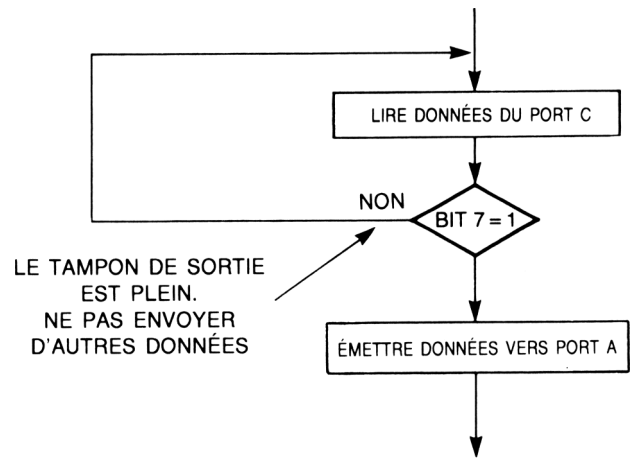


Figure 6.14 : *Ordinogramme montrant le déroulement des opérations nécessaires pour transférer des données vers le périphérique en employant les lignes du port A et les lignes de poids fort du port C.*

```
24E8              ODATA   EQU  24E8H      ; ADRESSES DES DONNEES DE SORTIE
;
;
1800                      CODE 1800H
;
1800  DB12         BACK    IN A,(12H)      ;LIRE DONNEES DU PORT C
1802  CB7F                 BIT 7,A         ;TESTER BIT 7 = 1
1804  CA0018               JP Z,BACK       ; PAS = 1, CONTINUEZ A LIRE PORT C
1807  3AE824               LD A,(ODATA)    ; METTRE DONNEES DE SORTIE DANS REG. A
180A  D310                 OUT (10H),A     ; EMETTRE DONNEES VERS PORT A
180C  C9                   RET
                           FIN
```

Figure 6.15 : *Programme Z80 pour réaliser l'ordinogramme de la Figure 6.14.*

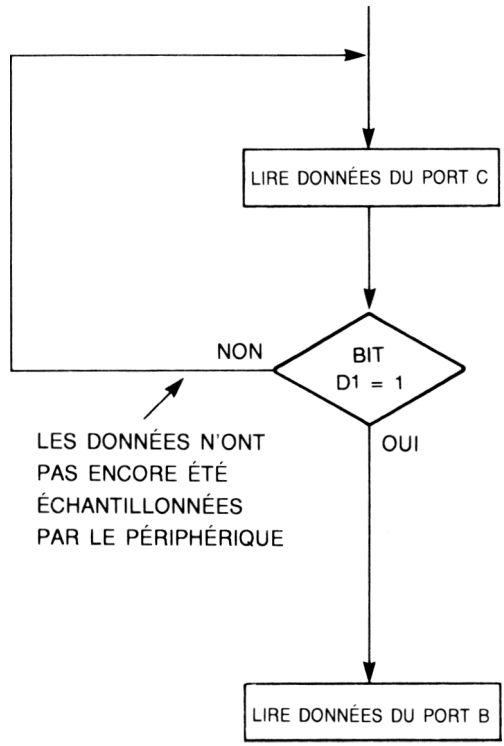


Figure 6.16 : *Ordinogramme montrant le déroulement des opérations nécessaires pour transférer dans le 8255 des données du périphérique via le port B et les lignes de poids faible du port C.*

```
;
;
1800   DB12      BACK    IN A,(12H)        ;LIRE PORT C
1802   CB4F              BIT 1,A           ;TEST BIT 1 = 1
1804   CA0018            JP Z,BACK         ;DONNEES PAS ENCORE ECHANTILLONNEE
1807   DB11              IN A,(11H)        ;DONNEES PRETES, LES LIRE
1809   C9                RET
                         FIN
```

Figure 6.17 : *Programme Z80 pour réaliser l'ordinogramme de la Figure 6.16.*

compte dans le 8255. La Figure 6.16 donne l'ordinogramme de ce type de transfert. Les mnémoniques permettant de réaliser l'ordinogramme apparaissent en Figure 6.17.

6.8 - Fonctionnement en mode 2 du 8255

Un autre mode de fonctionnement du 8255 est le mode 2. Dans ce mode, le dispositif peut utiliser le port A en port de données bidirectionnel ; c'est-à-dire que les huit lignes de données du port A peuvent transmettre des données vers le périphérique ou en recevoir de celui-ci. Quand le 8255 est programmé dans ce mode, le port A est représenté par un schéma tel que celui de la Figure 6.18.

La Figure 6.18 montre un *latch* de sortie et un *latch* d'entrée. Le *latch* de sortie stocke les données écrites dans le port par le CPU et attend l'autorisation du dispositif extérieur pour transférer les données dans les lignes de sortie du port A. Le *latch* d'entrée stocke les données envoyées dans les lignes d'entrée du port A par le dispositif extérieur.

Examinons maintenant le schéma de la Figure 6.18 et voyons comment sont transmises les données au dispositif extérieur. Tout d'abord le CPU écrit des données dans le *latch* de sortie du port A. La ligne de sortie, référencée $\overline{\text{OBF}}$ (*Output Buffer Full,* tampon de sortie plein) est alors mise à « vrai ». Ce signal informe électriquement les circuits extérieurs que les données venant du port A sont disponibles. Le signal $\overline{\text{OBF}}$ indique aussi au CPU que le tampon de sortie est plein et qu'il n'a pas été lu par le dispositif extérieur.

Le dispositif extérieur envoie ensuite un signal d'entrée $\overline{\text{ACK}}$ au 8255. Ce signal d'entrée autorise les données du *latch* de sortie à entrer dans les lignes de données du port A. Il réinitialise aussi la ligne $\overline{\text{OBF}}$, indiquant ainsi au CPU que les données de sortie du port A ont été lues. A ce moment-là, le CPU peut envoyer d'autres données au port de sortie A.

Avant que des données ne puissent être reçues par le 8255 en provenance du dispositif extérieur, celui-ci doit d'abord examiner l'état logique de la ligne de sortie IBF (*Input Buffer Full,* tampon d'entrée plein). Si cette ligne est au 1 logique, le tampon d'entrée est alors plein et les données d'entrée n'ont pas encore été lues par le microprocesseur. Supposons que la ligne IBF soit au 0 logique, indiquant qu'il n'y a pas de données dans le tampon.

Le dispositif extérieur va mettre les données dans les lignes de données du port A et activer l'entrée $\overline{\text{STB}}$ du 8255. Ceci va causer la prise en compte des données dans le *latch* interne du port A et mettre la ligne IBF à « vrai ». Le CPU va contrôler le niveau de la ligne IBF en lisant les données du port C. Quand la ligne IBF est à « vrai », les données sont prêtes à être lues. Quand le CPU lit les données, la ligne IBF va au 0 logique. A ce

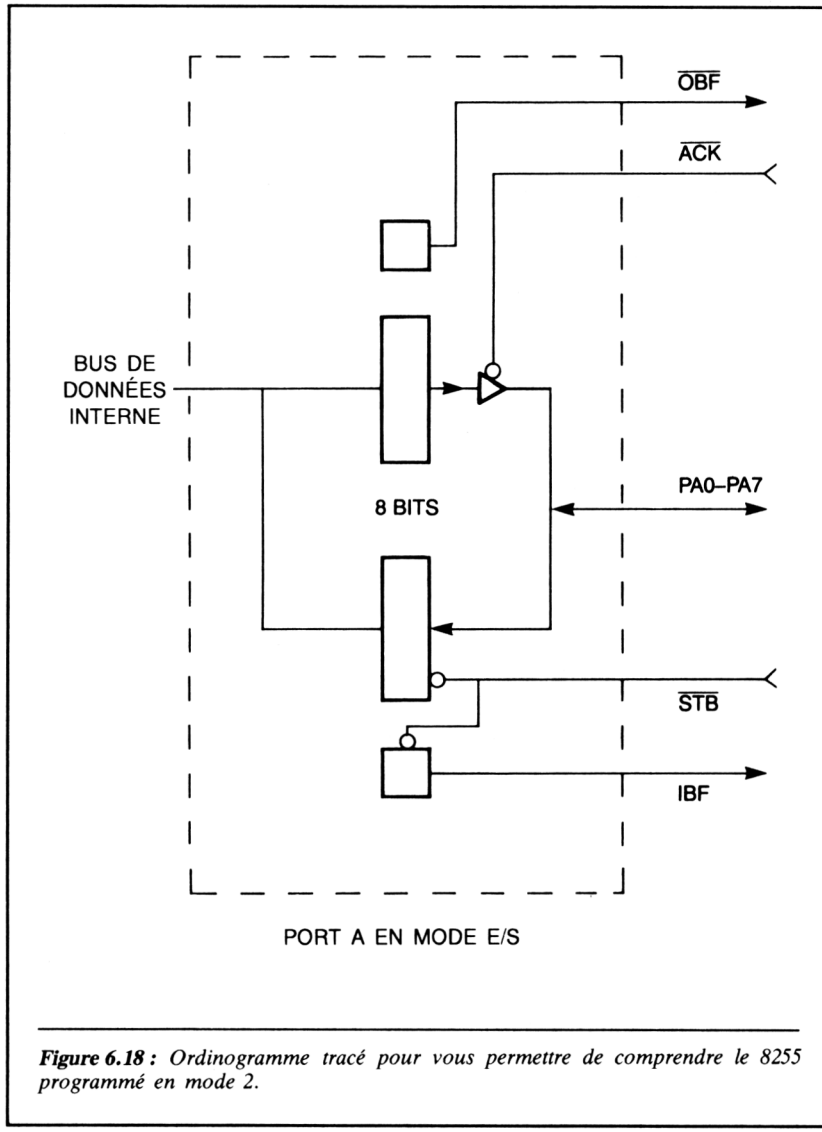


Figure 6.18 : *Ordinogramme tracé pour vous permettre de comprendre le 8255 programmé en mode 2.*

moment-là, le dispositif d'entrée-sortie extérieur peut envoyer d'autres données vers le 8255 de la façon qui vient d'être décrite.

Pour permettre au CPU d'examiner l'état des lignes d'interface extérieure, on emploie le port C pour refléter l'état logique de ces lignes. La Figure 6.19 donne la définition des bits du port C quand le 8255 est programmé en mode 2.

LIGNE DU PORT C	DEFINITION
PC0	I/O
PC1	I/O
PC2	I/O
PC3	$INTR_A$
PC4	$\overline{STB}_A$
PC5	IBF_A
PC6	$\overline{ACK}_A$
PC7	$\overline{OBF}_A$

Figure 6.19 : *Définition des broches du port C du 8255 en mode 2.*

RESUME

Dans ce chapitre, nous avons vu comment on emploie le boîtier 8255 avec le microprocesseur Z80. Nous en avons examiné l'organisation et montré comment il peut être relié aux bus du système à Z80. Enfin, nous avons examiné le logiciel nécessaire à son fonctionnement.

Le 8255 n'a pas été conçu à l'origine pour être employé avec le Z80. C'est cependant un boîtier d'interface extrêmement souple. Il peut simplifier les circuits servant à réaliser certaines applications. Comme il est d'un prix bas et facile à utiliser, vous pourrez très bien envisager son emploi comme moyen d'interface pour vos applications de Z80.

UTILISATION DU TEMPORISATEUR PROGRAMMABLE 8253

CHAPITRE 7

INTRODUCTION

Ce chapitre va nous montrer comment utiliser le boîtier temporisateur programmable 8253 avec le microprocesseur Z80. Nous allons commencer par examiner et décrire ce boîtier temporisateur. Nous l'interfacerons ensuite avec le microprocesseur Z80. Finalement, nous écrirons un logiciel permettant son emploi dans de nombreuses applications.

7.1 - Schéma du temporisateur programmable 8253

Commençons par examiner le schéma de la Figure 7.1 et par explorer les registres internes et les modes de fonctionnement de ce circuit intégré. Le schéma montre que le temporisateur a trois compteurs indépendants et programmables et qu'ils sont tous identiques. Dans ce chapitre, nous apprendrons comment appliquer et utiliser chaque compteur.

La Figure 7.1 montre aussi le *tampon du bus de données*. Ce bloc contient la logique qui effectue une adaptation entre le bus de données, allant vers le microprocesseur ou en provenance de celui-ci, et les registres internes du 8253. Il y a, en outre, le bloc référencé *logique de lecture-écriture* qui commande la lecture et l'écriture dans les registres des compteurs. Le dernier bloc - *le registre de mot de commande* - contient l'information programmée qui est envoyée au 8253 par le microprocesseur du système. Ce registre, en effet, détermine le fonctionnement logique du boîtier. La Figure 7.2 montre le brochage du 8253.

7.2 - Les trois lignes des compteurs : horloge, déclenchement et sortie

Chaque compteur du schéma de la Figure 7.1 a trois lignes qui lui sont reliées. Deux de ces lignes, l'horloge et le déclenchement, sont des entrées. La troisième, référencée *out* (sortie), est une sortie. La fonction de ces

PROGRAMMABLE INTERVAL TIMER

- **MCS-85™ Compatible 8253-5**
- **3 Independent 16-Bit Counters**
- **DC to 2 MHz**
- **Programmable Counter Modes**
- **Count Binary or BCD**
- **Single + 5V Supply**
- **24-Pin Dual In-Line Package**

The Intel® 8253 is a programmable counter/timer chip designed for use as an Intel microcomputer peripheral. It uses nMOS technology with a single +5V supply and is packaged in a 24-pin plastic DIP.

It is organized as 3 independent 16-bit counters, each with a count rate of up to 2 MHz. All modes of operation are software programmable.

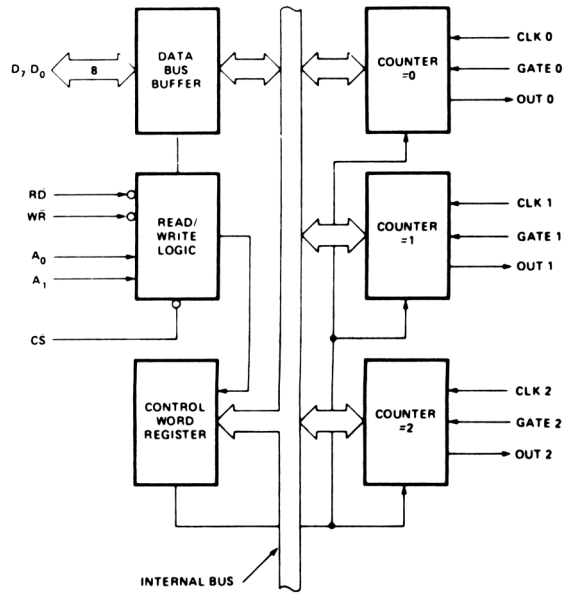


Figure 7.1 : *Schéma d'un temporisateur d'intervalle programmable 8253.*

lignes varie et dépend de la façon dont le boîtier est initialisé ou programmé. Voici une définition générale de ces lignes :

Horloge Cette entrée est l'entrée d'horloge du compteur. Ce compteur est à 16 bits. La fréquence maximale d'horloge correspond à 1/380 nanosecondes, soit 2,6 mégahertz. La fréquence d'horloge minimale correspond à une tension continue ou un fonctionnement statique.

Déclenchement Cette entrée peut agir en signal de déclenchement pour la ligne d'entrée d'horloge, ou bien elle peut agir en impulsion de départ, suivant le mode programmé du compteur.

Out Cette ligne de sortie distincte est le signal qui est la dernière sortie programmée du boîtier. Le fonctionnement effectif de la ligne de sortie dépend de la programmation du boîtier.

7.3 - Registres internes du 8253

Une liste des registres internes du 8253 apparaît en Figure 7.3. Examinons d'abord et étudions le *registre de mot de mode*. Ce registre détermine le fonctionnement global du boîtier. Les trois compteurs étant parfaitement indépendants, chacun d'eux peut être programmé en envoyant des données appropriées au registre de mot de mode. Nous allons montrer comment ceci peut être effectué au fur et à mesure de notre analyse. Définissons d'abord les quatre registres internes de la Figure 7.3.

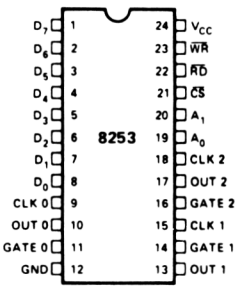


Figure 7.2 : *Brochage du temporisateur d'intervalle programmable.*

Registre de mot de commande Ce registre interne sert à y écrire des informations, avant l'utilisation du 8253. Ce registre est adressé quand les entrées A0 et A1 sont au 1 logique. Les données de ce registre commandent le mode de fonctionnement et la sélection du format de comptage, soit binaire, soit BCD (*Binary Coded Decimal*, décimal codé binaire). On peut seulement écrire dans ce registre. Le programmeur ne peut pas lire d'informations dans ce registre.

Compteurs n° 0, n° 1 et n° 2 Les compteurs sont identiques, chacun d'eux est constitué d'un décompteur prépositionnable à 16 bits. Ils sont complètement indépendants ; on peut les faire compter en BCD ou en binaire. Le contenu des compteurs peut être facilement lu par le microprocesseur. A la suite d'une lecture, les données des compteurs ne sont pas modifiées. Ceci permet au système de surveiller la valeur du compteur à tout moment, sans déranger le fonctionnement global du 8253.

	$\overline{RD}$	$\overline{WR}$	A0	A1	
COMPTEUR 0	1	0	0	0	CHARGER COMPTEUR 0
	0	1	0	0	LIRE COMPTEUR 0
COMPTEUR 1	1	0	0	1	CHARGER COMPTEUR 1
	0	1	0	1	LIRE COMPTEUR 1
COMPTEUR 2	1	0	1	0	CHARGER COMPTEUR 2
	0	1	1	0	LIRE COMPTEUR 2
MOT DE MODE OU	1	0	1	1	ÉCRIRE MOT DE MODE
MOT DE COMMANDE	0	1	1	1	PAS DE FONCTIONNEMENT

Figure 7.3 : *Liste des registres internes du 8253 qui vont programmer les compteurs internes.*

7.4 - Connexion du 8253 au Z80

Avant d'apprendre à programmer le 8253, apprenons la façon de le relier au microprocesseur Z80.

Le 8253 peut être considéré comme quatre ports d'entrée-sortie séparés. La sélection principale des ports d'entrée-sortie est obtenue par l'entrée $\overline{CS}$. Quand celle-ci est au 0 logique, le 8253 est mis en communication avec le Z80.

Les lignes d'adresses A0 et A1 déterminent quels ports d'entrée-sortie sont utilisés pour communiquer pendant le cycle d'entrée ou de sortie.

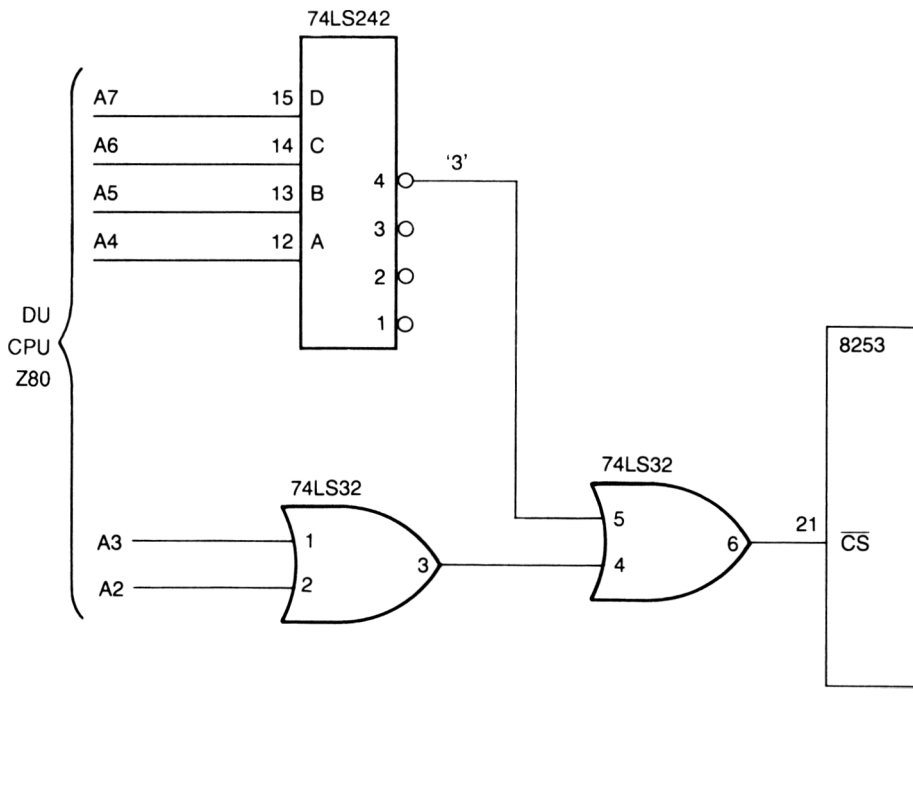


Figure 7.4 : *Schéma montrant comment l'entrée $\overline{CS}$ peut être décodée à partir des lignes d'adresses A7-A2.*

Cette structure d'entrée-sortie peut relier le dispositif aux ports d'entrée-sortie. On peut considérer le dispositif comme étant le 8253 et les ports qui lui sont associés sont les registres internes.

Le décodage de l'entrée $\overline{CS}$ du 8253 est effectué à partir des 6 bits de poids fort des lignes d'adresses de l'octet de poids faible, à savoir A7 à A2. Par exemple, plaçons le 8253 dans la structure d'entrée-sortie d'un système type. Nous supposerons que le 8253 est placé dans l'espace d'entrée-sortie 30 à 33. Autrement dit, les ports d'entrée-sortie 30H, 31H, 32H et 33H sont tous associés au 8253 (Voir Figure 7.4).

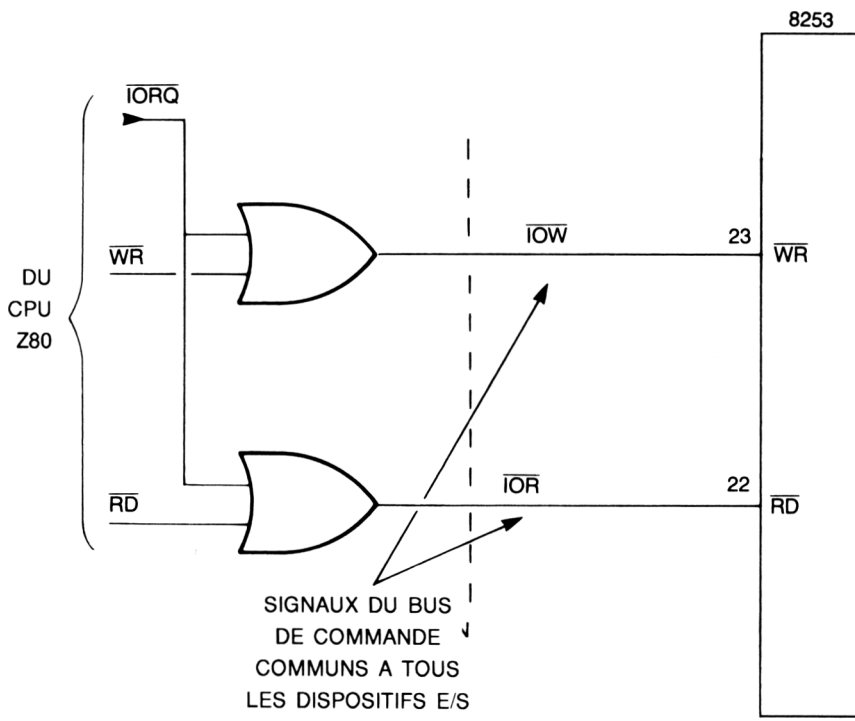


Figure 7.5 : *Schéma montrant comment sont produits les signaux de commandes $\overline{IOR}$ et $\overline{IOW}$. Remarquez la similitude avec la logique de production de signaux du Chapitre 6.*

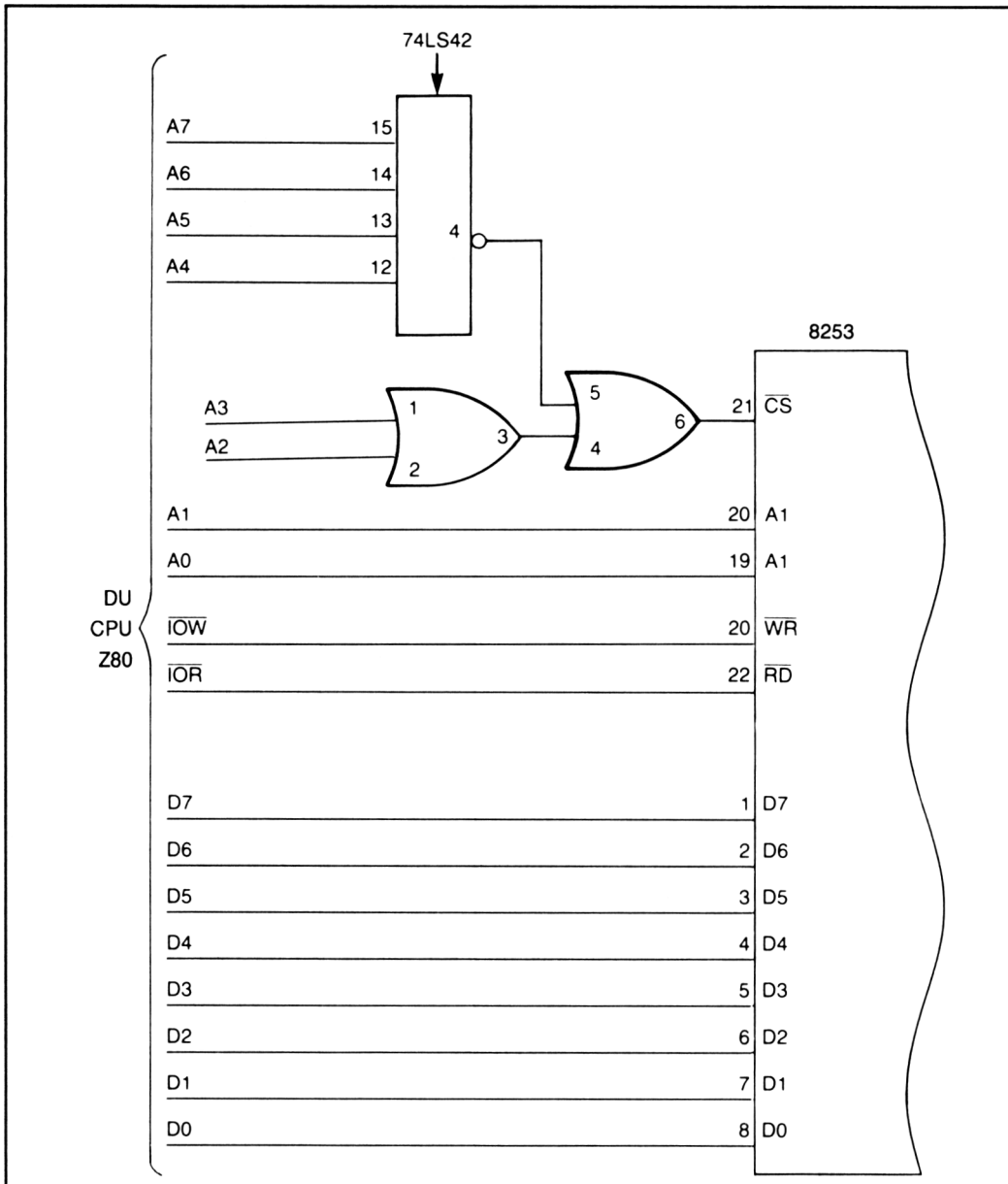


Figure 7.6 : *Vue complète des liaisons entre bus du 8253 et du Z80. Aucun tampon de données n'est utilisé dans ce cas.*

Les entrées $\overline{RD}$ et $\overline{WR}$ du 8253 sont connectées directement aux lignes de commandes du système, $\overline{IOR}$ et $\overline{IOW}$. Cette disposition est identique à celle employée pour le 8255 que nous avons vue au Chapitre 6. La Figure 7.5 montre cette connexion.

Les lignes de données du 8253 peuvent être reliées directement aux lignes du bus de données du Z80. (On suppose qu'il n'est pas nécessaire d'avoir ici des amplificateurs de données.) La Figure 7.6 montre la liaison complète du 8253 au Z80, avec données non amplifiées, tandis que la Figure 7.7 montre une liaison complète avec bus de données amplifié. L'amplificateur de bus de données employé est un 74LS245. Dans la Figure 7.7, chaque fois que le 8253 est choisi et que l'entrée $\overline{RD}$ est au 0 logique, le 74LS245 met les données du 8253 dans le bus de données du système à Z80. A tous les autres moments, le 74LS245 dirige les données provenant du bus de données du système à Z80 vers les broches d'entrées de données du 8253.

En employant les liaisons montrées dans les Figures 7.6 et 7.7, on peut établir une communication sûre entre le Z80 et le boîtier 8253. Il reste maintenant à programmer ce dernier de la façon appropriée à votre application.

7.5 - Programmation du 8253 (format du mot de commande)

Tous les modes de fonctionnement concernant les compteurs sont choisis en écrivant des octets dans le registre de commande. La Figure 7.8 montre le format du mot de commande. L'adresse du mot de commande est A0 = 1 et A1 = 1. Dans cette application de système, l'adresse du mot de commande est 33H.

Dans la Figure 7.8, les bits D7 et D6 sont référencés SC1 et SC0. Ces bits choisissent le compteur à programmer. Avant qu'un compteur ne soit programmé, il est nécessaire de définir, au moyen des bits de commande D7 et D6, quel compteur est à configurer. Il faut noter qu'un compteur, une fois configuré, le restera jusqu'à ce qu'il soit changé par un autre mot de commande. Les bits D7 et D6 sont définis comme suit :

D7	D6	COMPTEUR SELECTIONNE
0	0	0
0	1	1
1	0	2
1	1	valeur illégale

Les bits D5 et D4 du mot de commande de la Figure 7.8 sont définis comme mode lecture/chargement du registre choisi par les bits D7 et D6.

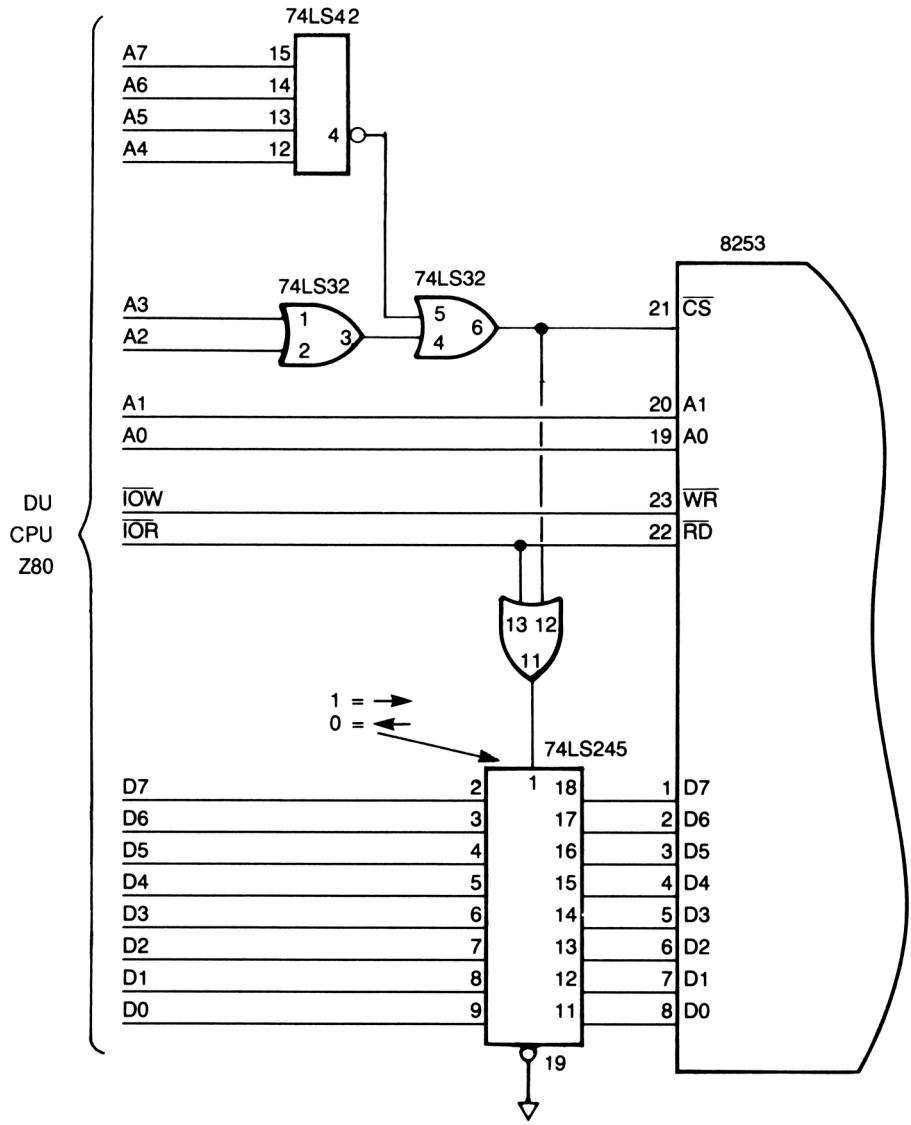


Figure 7.7 : Liaisons entre Z80 et 8253 employant des tampons de données.

OCTET DE COMMANDE D7-D0

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCP

Figure 7.8 : *Définition des bits du registre de commande du compteur.*

Les bits D5 et D4 définissent, pour le compteur choisi, la façon dont le microprocesseur doit lui lire ou écrire les données. Ces bits sont définis ainsi :

D5	D4	DEFINITION L/C
0	0	La valeur du compteur est *latchée*. Autrement dit, le compteur sélectionné voit son contenu transféré dans un *latch* temporaire, qui peut ensuite être lu par le CPU.
0	1	Lire/charger octet de poids faible seulement.
1	0	Lire/charger octet de poids fort seulement.
1	1	Lire/charger octet de poids faible d'abord, puis octet de poids fort.

Qu'est-ce que ces bits commandent aux circuits intégrés ? La première valeur 00H est le *mode latch de compteur*. Il sert à lire le compteur pendant son fonctionnement. Quand ce mode est spécifié, la valeur du compteur est *latchée* dans un registre interne au moment de l'opération d'écriture d'entrée-sortie du registre de commande. Quand survient une lecture de compteur, c'est cette valeur *latchée* qui est lue.

Si le mode *latch* n'est pas employé, il est alors possible que les données relues soient en train de changer pendant la lecture (Voir Figure 7.9). Ceci pourrait provoquer l'envoi de données erronées par le CPU. Pour lire la valeur du compteur pendant que celui-ci est encore en train de compter, il faut d'abord émettre un mot de commande de *latch,* puis émettre un autre mot de commande qui indique l'ordre des octets à lire.

Une autre méthode pour obtenir un compte stable de temporisation est d'interdire tout comptage pendant la lecture du registre. Cette technique est montrée en Figure 7.10. Chaque technique a ses inconvénients. La méthode *latch* risque de donner au microprocesseur une lecture « vieillie » de plusieurs cycles, suivant la vitesse de comptage et l'octet du compteur qui est lu. La fonction d'interdiction extérieure nécessite des circuits

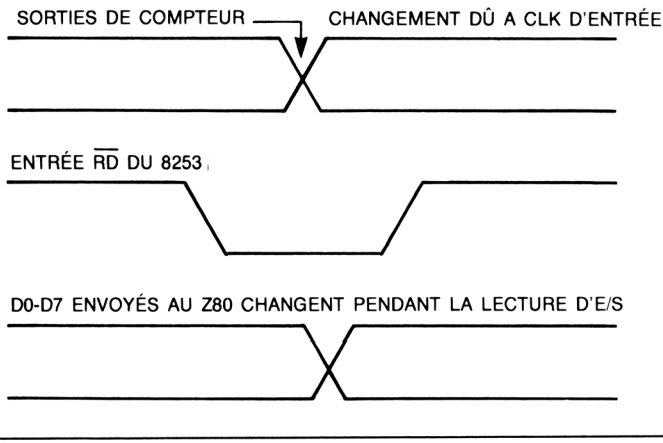


Figure 7.9 : *Diagramme des temps montrant comment les sorties de compteurs peuvent être en train de changer pendant que le microprocesseur lit le 8253. On peut éviter cela en latchant la valeur du compteur à sa sortie avant d'y lire la valeur.*

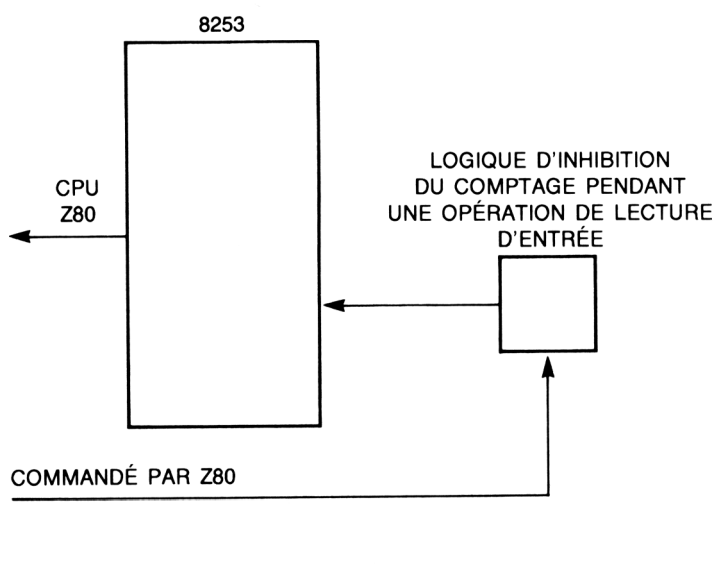


Figure 7.10 : *Schéma montrant comment une sortie de compteur peut être rendue stable pendant une opération de lecture par le CPU en inhibant d'une façon extérieure la ligne d'entrée d'horloge (de comptage).*

supplémentaires. En outre, elle risque de changer le fonctionnement global du système. C'est à vous de déterminer la meilleure façon de faire marcher le circuit dans une application donnée.

Les trois bits suivants du mot de commande de la Figure 7.8 sont D3, D2 et D1. Ils déterminent le mode fondamental de fonctionnement du compteur. Nous allons maintenant décrire le mode, puis continuer avec des exemples montrant comment utiliser le compteur dans chacun des six modes. Voici la description des modes :

D3	D2	D1	VALEUR DE MODE
0	0	0	mode 0 : interruption sur comptage final
0	0	1	mode 1 : monostable programmable
x	1	0	mode 2 : générateur d'impulsions
x	1	1	mode 3 : générateur de signaux carrés
1	0	0	mode 4 : déclenchement par logiciel
1	0	1	mode 5 : déclenchement par circuit

Le dernier bit du registre de commande D0 détermine comment le registre va compter, autrement dit, s'il va compter en BCD ou en binaire. Si D0 est au 1 logique, le comptage sera en BCD ; s'il est au 0 logique, le comptage sera en binaire. Les valeurs de comptage maximales, dans chaque mode de comptage, sont 2^{16} en binaire et 10^4 en BCD.

7.6 - Exemple de mode 0 : interruption à la fin du comptage

Voyons maintenant comment employer le mode 0 avec le 8253. Nous allons expliquer la fonction du mode 0 et définir les broches principales. Nous écrirons enfin un programme logiciel employant le mnémonique du Z80 pour mettre le 8253 en mode 0.

Ce mode permet au 8253 d'effectuer les opérations suivantes. Quand le compte interne est égal à 0, la broche OUT du compteur est mise au 1 logique. Ce compteur est programmé à une valeur initiale, puis il décompte à une vitesse donnée par la fréquence de l'horloge d'entrée. Quand le compte est égal à 0000, la broche OUT passe au 1 logique. Dans cette application, cette broche peut être connectée de façon à interrompre le microprocesseur. La sortie va rester au 1 logique, jusqu'à ce que le compteur soit rechargé avec le même compte ou un autre compte, ou bien jusqu'à ce qu'un mot de mode soit écrit dans le boîtier.

Une fois que le compteur commence à décompter, l'entrée de déclenchement peut interdire le comptage interne quand elle est au 0 logique. Pour bien voir comment fonctionne ce mode, prenons un exemple. Dans cet exemple, nous allons compter des événements avec le compteur n° 0. Ce

compte sera en BCD. Quand il atteindra 125, le Z80 sera interrompu. La routine de gestion d'interruption est à l'emplacement 0038H. Nous supposerons que le Z80 utilise le mode d'interruption 1. La Figure 7.11 donne le schéma de cet exemple.

Pour positionner le compteur, il faut programmer le mot de commande. En nous basant sur les attributions des bits de commande de la Figure 7.8, nous choisissons un mot de commande égal à 00110001. En voici l'explication :

```
Bits D7 et D6 = 00   Ceci indique compteur n° 0.
Bits D5 et D4 = 11   Ceci établit le registre pour charger
                     l'octet de poids faible d'abord et
                     l'octet de poids fort ensuite.
Bits de commande     Ceci définit le mode 0.
D3, D2 et D1 = 000
Enfin, D0 = 1        Ceci indique que le mode de comptage
                     sera BCD. Le mot de commande complet
                     écrit en port 33 est donc 00110001.
```

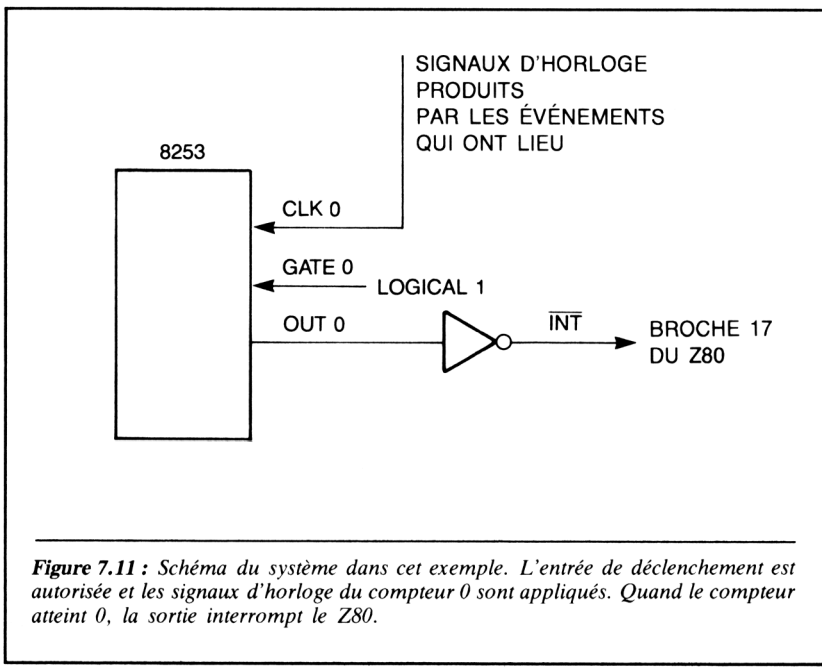


Figure 7.11 : *Schéma du système dans cet exemple. L'entrée de déclenchement est autorisée et les signaux d'horloge du compteur 0 sont appliqués. Quand le compteur atteint 0, la sortie interrompt le Z80.*

```
;
;
; ON SUPPOSE QUE LES INTERRUPTIONS SONT INHIBEES
;
0000   3E31              LD A,31H     ;ETABLIR MOT DE COMMANDE
0002   D333              OUT (33H),A  ;ENVOYER MOT DE COMMANDE AU 8253
0004   3E25              LD A,25H
0006   D330              OUT (30H),A  ;SORTIR LSB DU COMPTEUR 0
0008   3E01              LD A,01H
000A   D330              OUT (30H),A  ;SORTIR MSB DU COMPTEUR 0
;
;  LE COMPTE EST EN BCD
;
;       LE COMPTAGE DEMARRE QUAND MSB EST SORTI
;
000C   FB                EI
                         END
```

Figure 7.12 : *Programme Z80 pour initialiser le 8253.*

```
;
;   ON SUPPOSE QUE LE Z80 UTILISE LES INTERRUPTIONS
;   EN MODE 1
;
;   LE COMPTE A ATTEINT 125, A CE MOMENT LA
;   ON RECHARGE LE COMPTEUR N° 0
;
0038    3E25             LD A,25H
003A    D330             OUT (30H),A    ;CHARGER LSB
003C    3E01             LD A,01H
003E    D330             OUT (30H),A    ;CHARGER MSB
;
;  UN NOUVEAU DECOMPTAGE EST DEMARRE
;
0040    FB               EI
0041    C9               RET
;
; UNE PARTIE DE LA ROUTINE D'INTERRUPTION SERAIT
; D'EFFECTUER UNE PROCEDURE DUE A L'INTERRUPTION
; QUI S'EST PRODUITE. DANS CET EXEMPLE ON SE CONTENTE
; DE REINITIALISER LE COMPTEUR ET DE DEMARRER
;
                         END
```

Figure 7.13 : *Routine de gestion d'interruption du Z80 utilisant des interruptions en mode 1. Le 8253 fonctionne en mode 0.*

Il faut ensuite écrire le compte initial dans le registre du compteur. Comme nous comptons en BCD, l'octet de poids faible sera 35 en décimal et l'octet de poids fort 01. Ces deux octets sont écrits dans le port 30. La Figure 7.12 montre une partie du listing du Z80 qui initialise le 8253.

Dès que le second octet de données est chargé dans le registre 0, le comptage démarre. Quand le compte atteint 125, le registre du compteur atteint 0 et la broche OUT passe au 1 logique, interrompant ainsi le Z80. La routine de gestion d'interruption va faire le nécessaire pour cette application. Une partie de la routine de gestion d'interruption réinitialise le 8253. La Figure 7.13 donne un exemple de routine de gestion d'interruption utilisant les interruptions en mode 1 du Z80. Cette routine a lieu avec le 8253 en mode 0.

7.7 - Mode 1 : monostable programmable

En mode 1, le 8253 peut donner une impulsion de sortie correspondant à un nombre entier d'impulsions d'horloge. Le monostable est déclenché sur le front montant de l'entrée de déclenchement. Si le déclenchement survient pendant la sortie d'impulsion, le dispositif est redéclenché. Ceci est montré par la Figure 7.14.

Supposons que la fréquence d'horloge d'entrée soit d'un MHz. Nous voulons programmer le 8253 pour avoir une impulsion de sortie ayant exactement 75 microsecondes. Le programme Z80 le permettant apparaît en Figure 7.15. On a supposé, là, que le compteur 1 est utilisé en monostable programmable.

7.8 - Mode 2 : générateur d'impulsions

En mode 2, le 8253 devient un compteur « diviseur par n ». La broche de sortie du compteur va au niveau bas pendant une période de l'horloge d'entrée. Le temps écoulé entre chaque aller à 0 des impulsions de sortie dépend du compte existant à ce moment-là dans le registre du compteur. Voir Figure 7.16. Si le compteur est chargé entre les impulsions de sortie, la période en cours ne sera pas modifiée. Une nouvelle période aura lieu pendant la séquence de comptage suivante.

A titre d'exemple, supposons que nous voulions obtenir, avec le compteur n° 2, une fréquence de sortie égale à 638 Hz, donc une période de 1/638 soit 1,567 millisecondes ou 1.567 microsecondes. Si l'on applique un signal d'horloge de 1 mégahertz à l'entrée d'horloge du compteur 2, celui-ci devra alors être programmé à 1567. Ceci peut être effectué en BCD ou en décimal. La Figure 7.17 donne un exemple de programme Z80 pour une opération telle que celle-ci.

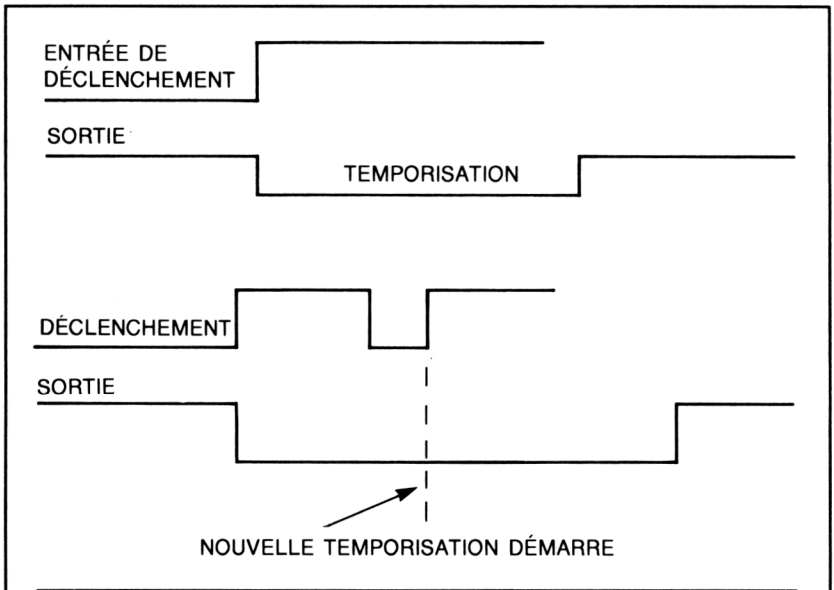


Figure 7.14 : *Diagramme des temps montrant comment le 8253 est employé en monostable programmable. Quand l'entrée de déclenchement va au 1 logique, la sortie va au 0 logique pendant le temps programmé du compteur. Si un nouveau signal de déclenchement survient avant la fin du comptage, le comptage est redéclenché.*

```
;
;
;
0000   3E72            LD A, 01110010B
0002   D333            OUT (33H),A        ;SORTIE MOT DE COMMANDE
;                                          VERS 8253
; CTR 1, RL = 3, M=1, BINARY COUNT
;
0004   3E4B            LD A, 75           ;DECIMAL 75
0006   D331            OUT (31H),A        ;SORTIR VERS CTR1 POIDS FAIBLE
0008   3E00            LD A, 00
000A   D331            OUT (31H),A        ;SORTIR VERS CTR1 POIDS FORT
;
;
;  LE BOITIER EST MAINTENANT PROGRAMME POUR ATTENDRE
;  UN DECLENCHEMENT DU COMPTEUR
;
                       END
```

Figure 7.15 : *Programme Z80 pour emploi du 8253 en monostable programmable.*

7.9 - Mode 3 : générateur de signaux carrés

Le mode 3 est similaire au mode 2 excepté que la sortie est à l'état haut pendant une moitié de la période et à l'état bas pendant l'autre moitié. Si le compte est impair, la sortie sera à l'état haut pendant (n+1)/2 et à l'état bas pendant (n−1)/2. La Figure 7.18 montre un programme Z80 qui configure le compteur n° 0 pour obtenir un signal carré de fréquence 10 KHz, à partir d'une fréquence d'horloge de 1 mégahertz.

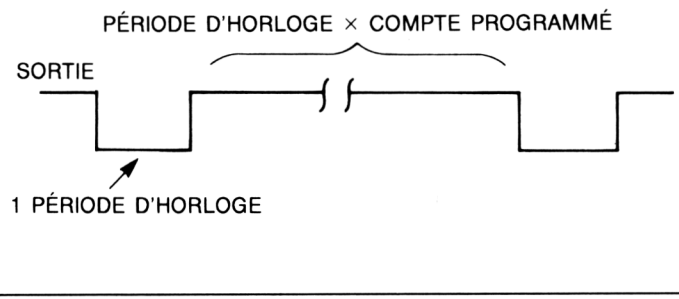


Figure 7.16 : *Diagramme des temps montrant la configuration de sortie du 8253 en générateur d'impulsions, en mode 2. La sortie va au 0 logique pendant une période de signal d'horloge. Le temps entre impulsions est déterminé par le compte qui a été programmé dans le 8253.*

```
;
;
0000   3EB5            LD A,10110101B
0002   D333            OUT (33H),A          ;SORTIR MOT DE COMMANDE
0004   3E67            LD A,67H
0006   D332            OUT (32H),A          ;CTR2, LSB EN BCD
0008   3E15            LD A,15H
000A   D332            OUT (32H),A          ;CTR2, MSB EN BCD
;
;
;  LE BOITIER EST MAINTENANT INITIALISE ET LA FREQUENCE DE SORTIE
;  SERA 638 HZ AVEC UNE FREQUENCE D'ENTREE DE 1 MHZ
;
                       END
```

Figure 7.17 : *Programme Z80 pour emploi du 8253 en générateur d'impulsions. La fréquence de sortie est de 638 Hz. La fréquence d'horloge est de 1 MHz.*

```
;
;
;
0000    3E36            LD A,00110110B
0002    D333            OUT (33H),A         ;ETABLIR MOT DE COMMANDE
0004    3E64            LD A, 100           ;COMPTEUR 0
0006    D330            OUT (30H),A         ;LSB DU CTRO EN BINAIRE
0008    3E00            LD A, 00H
000A    D330            OUT (30H),A         ;MSB DU CTRO EN BINAIRE
;
;
; LE BOITIER EST MAINTENANT INITIALISE ET FONCTIONNE
; A UNE FREQUENCE DE 10 KHZ AVEC UNE FREQUENCE D'ENTREE DE 1 MHZ
;
                        END
```

Figure 7.18 : *Programme Z80 pour utiliser le 8253 en générateur de signaux carrés. La fréquence de sortie est de 10 KHz. La fréquence d'horloge d'entrée est de 1 MHz.*

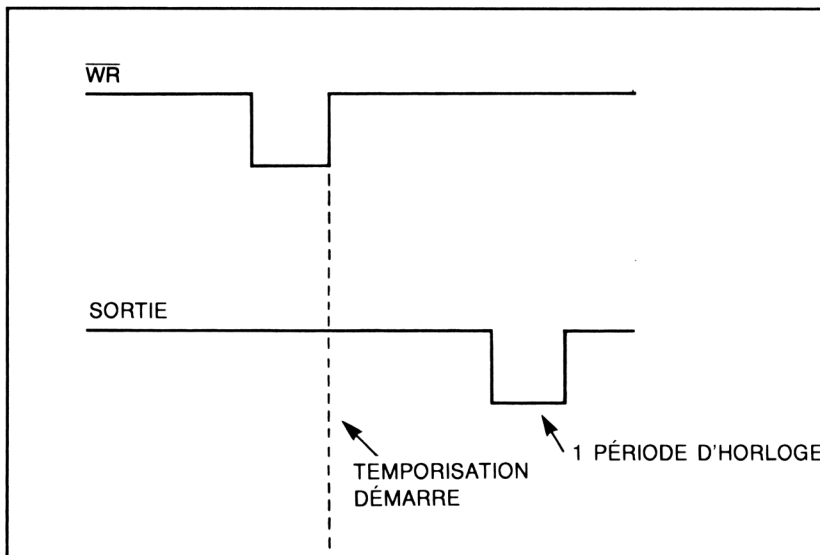


Figure 7.19 : *Diagramme des temps montrant comment la sortie du 8253 apparaît quand ce boîtier est employé en échantillonneur programmable déclenché par logiciel. Quand le Z80 effectue une opération d'écriture entrée-sortie, le temporisateur démarre. A la fin du comptage, la sortie va à l'état bas pendant une période d'horloge.*

7.10 - Mode 4 : échantillonnage déclenché par logiciel

Dans ce mode, le programmeur peut établir le compteur pour obtenir un délai qui débute quand le registre du compteur est chargé. A la fin du comptage, quand le compteur arrive à 0, la sortie passe à l'état bas pendant une période d'horloge puis repasse à l'état haut. Ceci est montré en Figure 7.19. Quand ce mode est établi, la sortie va à l'état haut.

7.11 - Exemple

A titre d'exemple, programmons le 8253 pour qu'il donne une temporisation de 100 millisecondes déclenchée par logiciel à la sortie du compteur n° 2. Nous supposons que la fréquence d'entrée est d'un mégahertz, ce qui nécessite un comptage d'entrée de 10^5. Nous voulons, en outre, que ce comptage soit en BCD. Le comptage BCD maximum d'un registre est égal à 10^4. Nous utiliserons donc deux compteurs. Le premier divisera la fréquence pour l'amener à 1 kilohertz. Le second fournira l'échantillonnage déclenché par logiciel. Ceci est montré en Figure 7.20. La Figure 7.21 présente un programme de Z80 qui utilise le 8253 de la façon expliquée dans cet exemple.

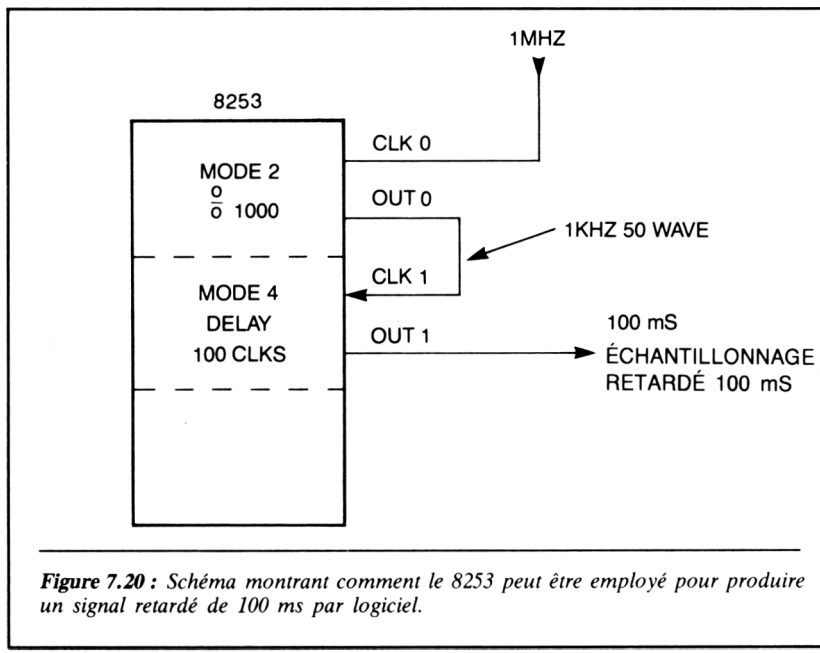


Figure 7.20 : *Schéma montrant comment le 8253 peut être employé pour produire un signal retardé de 100 ms par logiciel.*

```
0000    3E35            LD A,00110101B
0002    D333            OUT (33H),A         ;ETABLIR MOT DE COMMANDE
0004    3E00            LD A,00H            ;DU COMPTEUR 0
0006    D330            OUT (30H),A         ;CHARGER LSB DU CTR0
0008    3E10            LD A,10H
000A    D330            OUT (30H),A         ;CHARGER MSB DU CTR0
;
;
;   LE COMPTEUR 0 EST INITIALISE POUR UNE FREQUENCE DE SORTIE DE 1 KHZ,
;   EN SIGNAL CARRE, AVEC UNE FREQUENCE D'ENTREE DE 1 MHZ.
;   CETTE SORTIE EST CONNECTEE A L'ENTREE D'HORLOGE DU CTR1.
;
000C    3E79            LD A,01111001B
000E    D333            OUT (33H),A         ;MOT DE COMMANDE DU CTR1
0010    3E00            LD A,00H
0012    D331            OUT (31H),A         ;LSB DU CTR1
0014    3E01            LD A,01H
0016    D331            OUT (31H),A         ;MSB DU CTR1
;
;   LA DERNIERE SORTIE VERS CTR1 N'A PAS BESOIN D'ETRE EFFECTUEE
;   TANT QUE LE DELAI D'ATTENTE N'EST PAS DEMANDE.
;
                        END
```

Figure 7.21 : *Programme de Z80 d'initialisation du 8253 pour faire fonctionner un échantillonnage de 100 ms par logiciel. Ce programme utilise le schéma de la Figure 7.20.*

7.12 - Mode 5 : échantillonnage déclenché par circuit

Quand le compteur est dans ce mode, le front montant de l'entrée de déclenchement provoque le démarrage du compteur. La sortie va à l'état bas pendant une période d'horloge à la fin du comptage. Le dispositif est redéclenchable, ce qui veut dire que si l'entrée de déclenchement passe du niveau bas au niveau haut pendant une séquence de comptage, la séquence va redémarrer (Voir Figure 7.22). La Figure 7.23 montre un programme Z80 qui initialise le compteur n° 1 du 8253 afin de l'employer comme échantillonneur déclenché par circuit.

7.13 - Utilisation des broches d'entrée de déclenchement

Chaque mode de fonctionnement des compteurs utilise différemment la broche d'entrée de déclenchement. Le tableau de la Figure 7.24 résume le fonctionnement de cette commande.

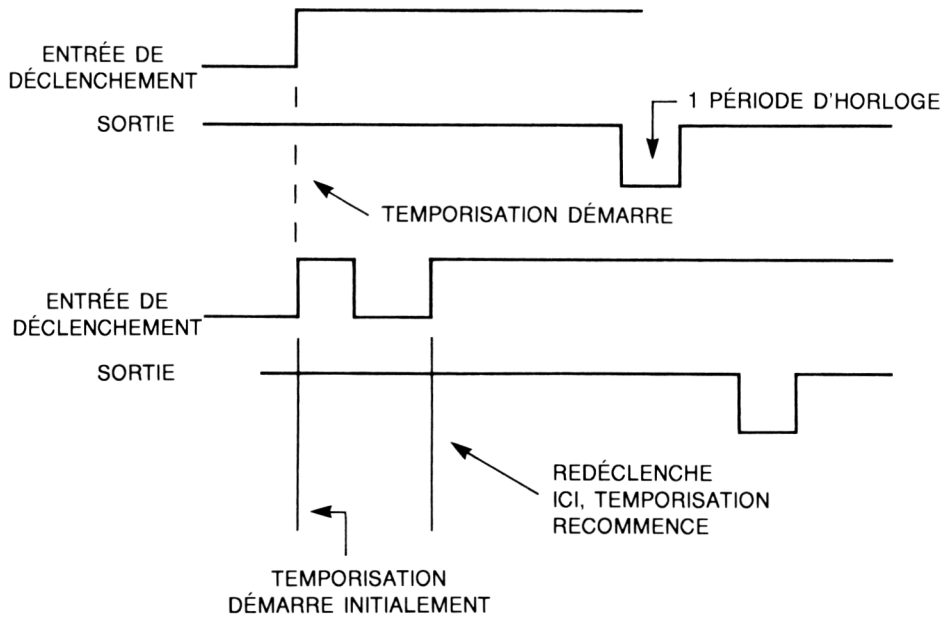


Figure 7.22 : *Diagramme des temps montrant comment fonctionne le 8253 en échantillonneur déclenché par circuit. Quand le signal d'entrée de déclenchement extérieur va au 1 logique, le temporisateur commence à fonctionner. Si un déclenchement extérieur se produit à nouveau avant la fin du fonctionnement de la temporisation, le temporisateur se redéclenche.*

```
;
;
;
;
0000    3E7A            LD A,01111010B
0002    D333            OUT (33H),A         ;MOT DE COMMANDE DU CTR1
0004    3E55            LD A,85
0006    D331            OUT (31H),A         ;SORTIE 85 BINAIRE VERS CTR1
0008    3E00            LD A, 00H           ;POIDS FAIBLE
000A    D331            OUT (31H),A         ;SORTIE 00 VERS CTR1 POIDS FORT
;
;   LE CTR EST MAINTENANT INITIALISE POUR ATTENDRE
;   UNE ENTREE DE DECLENCHEMENT EXTERNE SUR TRG1
;
                        END
```

Figure 7.23 : *Programme de Z80 pour initialiser et utiliser le 8253 en mode échantillonnage déclenché par circuit suivant le diagramme des temps de la Figure 7.22.*

Etat du signal / Modes	Bas ou allant vers le bas	Montant	Haut
0	Inhibe comptage	— —	Autorise comptage
1	— —	1) Initialise comptage 2) Remet sortie à 0 après horloge suivante	— —
2	1) Inhibe comptage 2) Met sortie immédiatement haute	1) Recharge compteur 2) Initialise comptage	Autorise comptage
3	1) Inhibe comptage 2) Met sortie immédiatement haute	Initialise comptage	Autorise comptage
4	Inhibe comptage	— —	Autorise comptage
5	— —	Initialise comptage	— —

Figure 7.24 : *Tableau montrant les différentes utilisations de la commande d'entrée de déclenchement du 8253.*

RESUME

Nous avons examiné dans ce chapitre les points importants concernant l'emploi du temporisateur programmable avec le CPU Z80. Nous avons donné un schéma du 8253 et une analyse de chaque mode de fonctionnement. Le 8253 peut être employé pour servir à la plupart des fonctions de temporisation dans de nombreuses applications utilisant le Z80 pour commander un système. Les exemples de ce chapitre ont montré que l'emploi de ce boîtier ne constitue pas une tâche complexe.

UTILISATION DU PIO DU Z80

CHAPITRE 8

INTRODUCTION

Dans ce chapitre, nous allons montrer comment le Mostek MK3881 (PIO du Z80) peut être utilisé, associé au microprocesseur Z80, comme dispositif d'entrée-sortie programmable. Contrairement au 8255, dispositif d'entrée-sortie parallèles examiné au Chapitre 6, le Mostek MK3881 a été spécialement conçu pour être employé avec le microprocesseur Z80.

Le PIO du Z80 est un dispositif d'entrée-sortie possédant différents modes de fonctionnement. Nous apprendrons dans ce chapitre comment fonctionne chacun de ces modes et comment les utiliser avec le microprocesseur Z80 dans la plupart des systèmes.

8.1 - Schéma du PIO

Nous allons commencer notre analyse en examinant le schéma de la Figure 8.1. Ce schéma montre que le PIO contient deux ports d'entrée-sortie indépendants : A et B. Nous verrons plus loin que ces ports sont presque identiques au point de vue caractéristiques de fonctionnement.

A la sortie de chaque port, il y a huit lignes de données et deux lignes de commandes. Les lignes de sortie et d'entrée de port sont compatibles TTL. Chaque sortie peut conduire 2,0 milliampères à l'état logique 0 et 250 microampères à l'état logique 1. C'est une capacité suffisante pour un dispositif TTL standard qui nécessite 1,6 milliampères à l'état logique 0 et 40 microampères à l'état logique 1.

En outre, la Figure 8.1 montre deux autres blocs : la logique de commande interne et la commande d'interruption. La *logique de commande interne* dirige les données internes du dispositif vers le registre interne voulu au moment approprié. La *commande d'interruption* permet au PIO de demander des interruptions d'une façon adéquate et de prendre en compte les accusés de réception d'interruptions provenant du microprocesseur Z80. Nous consacrerons plus tard toute une section de ce chapitre à vous montrer comment on peut programmer le PIO pour qu'il manipule les fonctions d'interruption.

8.2 - Brochage du PIO du Z80

La Figure 8.2 montre le brochage du PIO du Z80 en boîtier DIP à 40 broches. L'alimentation est fournie par le +5v et la masse. Examinons maintenant les principales broches du PIO et cherchons à comprendre ce que peut faire chacune d'elles. Nous vous montrerons plus tard comment on peut connecter effectivement le PIO au microprocesseur Z80 pour obtenir une liaison efficace.

Examinons maintenant en détail le brochage de la Figure 8.2 :

D7-D0 Ce sont les lignes d'entrée et de sortie de données qui reçoivent ou envoient des informations au microprocesseur Z80.

B/A Select (Sélection B/A) Cette ligne d'entrée distincte détermine si les données du bus de données sont en communication avec le port B (1 logique) ou avec le port A (0 logique). Cette ligne d'entrée est généralement connectée au bit A0 des lignes d'adresses du système.

C/D Select (Sélection Commande/Données) La valeur logique de cette entrée indique au PIO si les données en cours d'écriture dans le PIO

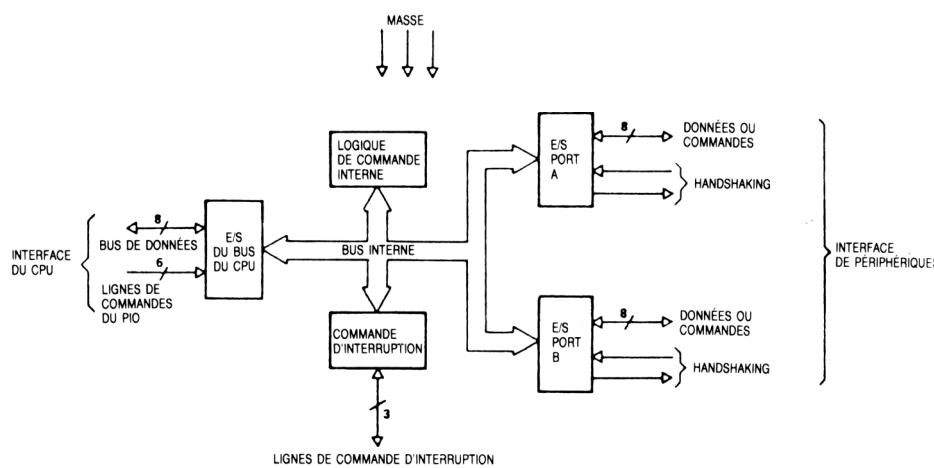


Figure 8.1 : Schéma du PIO du Z80.

constituent réellement des données ou bien un mot de commande. Les mots de commande servent à programmer le PIO dans les différents modes de fonctionnement.

$\overline{\mathbf{CE}}$ *(Chip Enable input*, entrée de validation de boîtier) Chaque fois que cette ligne est au 0 logique actif, les données peuvent être écrites ou lues dans le dispositif d'entrée-sortie. L'entrée $\overline{CE}$ est habituellement un espace d'adresse décodé à partir des huit bits de poids faible du bus d'adresses du système.

CLOCK (horloge) Une entrée d'horloge monophasée est employée ici. Le PIO utilise l'horloge pour synchroniser certaines opérations internes. (Ces opérations seront développées un peu plus loin.)

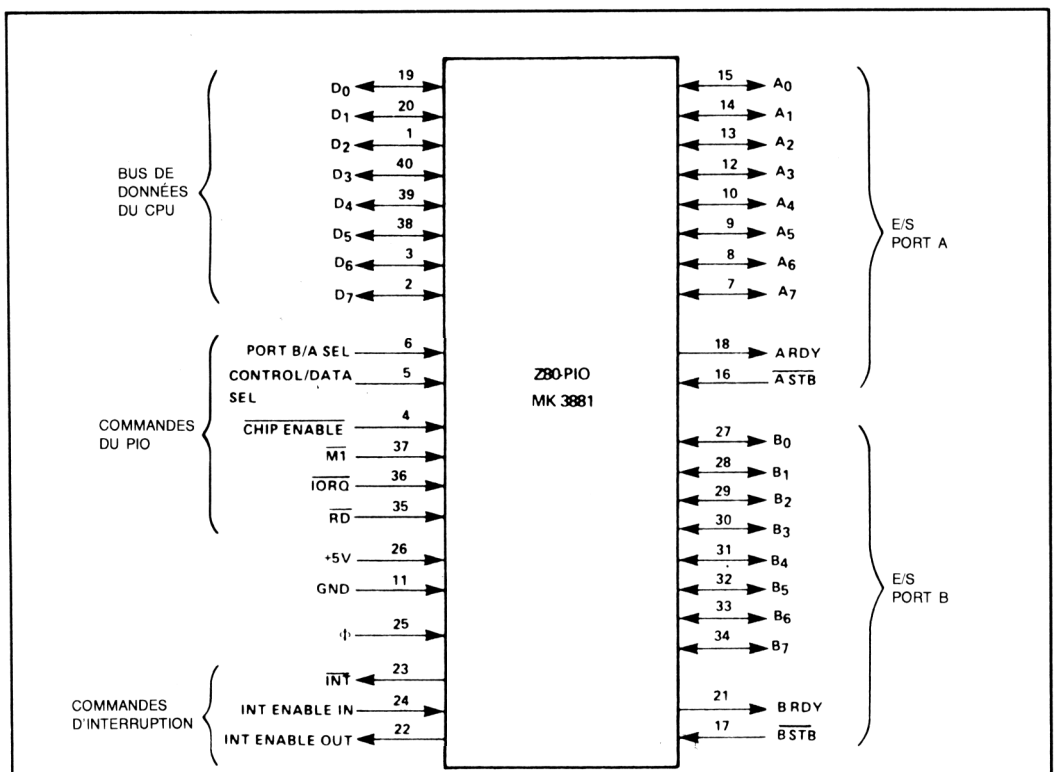


Figure 8.2 : *Brochage du boîtier DIP à 40 broches du PIO du Z80.*

$\overline{M1}$ C'est la sortie $\overline{M1}$ du Z80. Le PIO utilise cette ligne d'entrée pour commander plusieurs fonctionnements internes. Quand $\overline{M1}$ et $\overline{RD}$ sont tous les deux actifs, le Z80 est en train de rechercher un code opération. Il est important pour le PIO de reconnaître cet état parce que le dispositif reconnaîtra automatiquement un certain code opération. C'est l'instruction RETI (retour d'interruption). Le signal $\overline{M1}$ sert aussi à permettre au PIO de répondre à l'accusé de réception d'interruption provenant du Z80. Rappelez-vous que, quand $\overline{IORQ}$ et $\overline{M1}$ sont actifs, le Z80 est en train d'accuser réception d'une interruption.

En plus de ces deux opérations, le signal $\overline{M1}$ va aussi effectuer les suivantes :

- Il va synchroniser la logique d'interruption du PIO.
- Quand $\overline{M1}$ a lieu sans $\overline{RD}$ ou $\overline{IORQ}$ actifs, le PIO est mis en état de réinitialisation interne. Ceci est analogue à une réinitialisation du boîtier par circuit. Remarquez qu'il n'y a pas de broche d'entrée de réinitialisation extérieure sur le boîtier.

$\overline{IORQ}$ (*Input Output ReQuest*, demande d'entrée-sortie) C'est une entrée du PIO qui se connecte directement à la sortie $\overline{IORQ}$ du microprocesseur Z80. Ce signal est employé avec B/A select, C/D select et $\overline{CE}$ pour transférer commandes et données entre Z80 et PIO. Quand l'entrée $\overline{IORQ}$ et l'entrée $\overline{CE}$ sont toutes deux actives (0 logique), le PIO communique électriquement avec le Z80.

Une autre fonction de l'entrée $\overline{IORQ}$ est d'informer le PIO quand un accusé de réception d'interruption est en cours d'exécution. Quand $\overline{IORQ}$ et $\overline{M1}$ sont tous deux au 0 logique, le Z80 est en train d'accuser réception d'une interruption. Si le PIO est programmé pour répondre à des interruptions et constitue, en fait, le dispositif d'interruption, un vecteur d'interruption va être mis automatiquement dans le bus de données du système. Nous montrerons, plus loin dans ce chapitre, comment cela peut être accompli.

$\overline{RD}$ (*ReaD*, lecture) Cette entrée est connectée directement à la ligne de sortie $\overline{RD}$ du Z80. Rappelez-vous les premiers chapitres sur les opérations du Z80 où l'on expliquait que la sortie $\overline{RD}$ est au 0 logique chaque fois que le microprocesseur est en train de lire les données en mémoire ou en entrée-sortie. Par conséquent, quand l'entrée $\overline{IORQ}$ du PIO est active et que les entrées $\overline{RD}$ et $\overline{CE}$ le sont aussi, le Z80 lit les données dans le PIO.

Remarquez que le PIO n'a pas d'entrée $\overline{WR}$ (*WRite*, écriture). Si $\overline{IORQ}$ et $\overline{CE}$ sont tous deux actifs, mais que $\overline{RD}$ n'est pas actif, le Z80 doit écrire dans le PIO. L'opération d'écriture est une opération par défaut et non explicitement spécifiée par une ligne d'entrée $\overline{WR}$. Une telle ligne serait, en effet, redondante.

IEI et IEO Ce sont les lignes *Interrupt Enable Input* et *Output* (entrée et sortie d'autorisation d'interruption). On les utilise dans le chaînage de priorité d'interruption du Z80. Nous les étudierons en détail dans la section de ce chapitre concernant les interruptions.

$\overline{\text{INT}}$ Cette sortie est une connexion à drain ouvert, active chaque fois que le PIO a été programmé pour des interruptions et est en train de demander activement une interruption au microprocesseur Z80. $\overline{\text{INT}}$ sera expliqué dans une section ultérieure sur les interruptions.

A0-A7 Ce sont les lignes d'entrée-sortie du port A. C'est par ces lignes que le PIO communique avec un périphérique. A0 est le bit de poids faible de ce bus d'entrée-sortie.

$\overline{\text{ASTB}}$ Cette entrée reçoit les signaux d'un périphérique pour effectuer une procédure de *handshaking* dans le registre du port A. Les caractéristiques de fonctionnement effectives de cette ligne d'entrée dépendent du mode de fonctionnement du PIO. Voici quelques utilisations possibles de cette entrée :

- Quand le port est dans le mode « sortie seulement », le front montant de ce signal indique que le périphérique a reçu les données.
- Dans le mode « entrée seulement », le signal est employé par le périphérique pour charger les données provenant du périphérique dans le registre du port A. Les données sont chargées dans le registre interne du PIO, quand cette entrée est active. Nous montrerons qu'il n'est pas nécessaire d'employer cette ligne d'entrée pour utiliser le PIO comme port d'entrée.
- Quand le port A est programmé pour le fonctionnement en mode bidirectionnel, les données provenant du registre de sortie interne A sont mises dans les lignes A0-A7. Quand la ligne va au 1 logique, les données de sortie sont supprimées et la réception de données par le périphérique fait l'objet d'un accusé de réception. Nous expliquerons ceci plus en détail ultérieurement.

ARDY (*register A ReaDY*, registre A prêt) La fonction de cette sortie dépend du mode de fonctionnement du PIO. Voici quelques fonctions possibles, semblables à celles de $\overline{\text{ASTB}}$:

- Quand le registre A est programmé en mode de sortie, cette ligne indique au périphérique que les données ont été chargées dans les lignes A0-A7. Le périphérique connecté aux lignes du port A utilise cette sortie qui indique que les données peuvent être prises ou lues dans le port.

- Quand le port est programmé en mode « entrée seulement », ce signal indique que les données d'entrée ont été lues dans le registre d'entrée du port A par le CPU du Z80.

- Quand le port est programmé en mode bidirectionnel, le signal est actif quand les données sont disponibles dans le registre de sortie du port A. Ce signal indique au périphérique que des données sont présentes. Le périphérique doit alors activer la ligne $\overline{\text{ASTB}}$ afin de placer les données de sortie dans les lignes A0-A7.

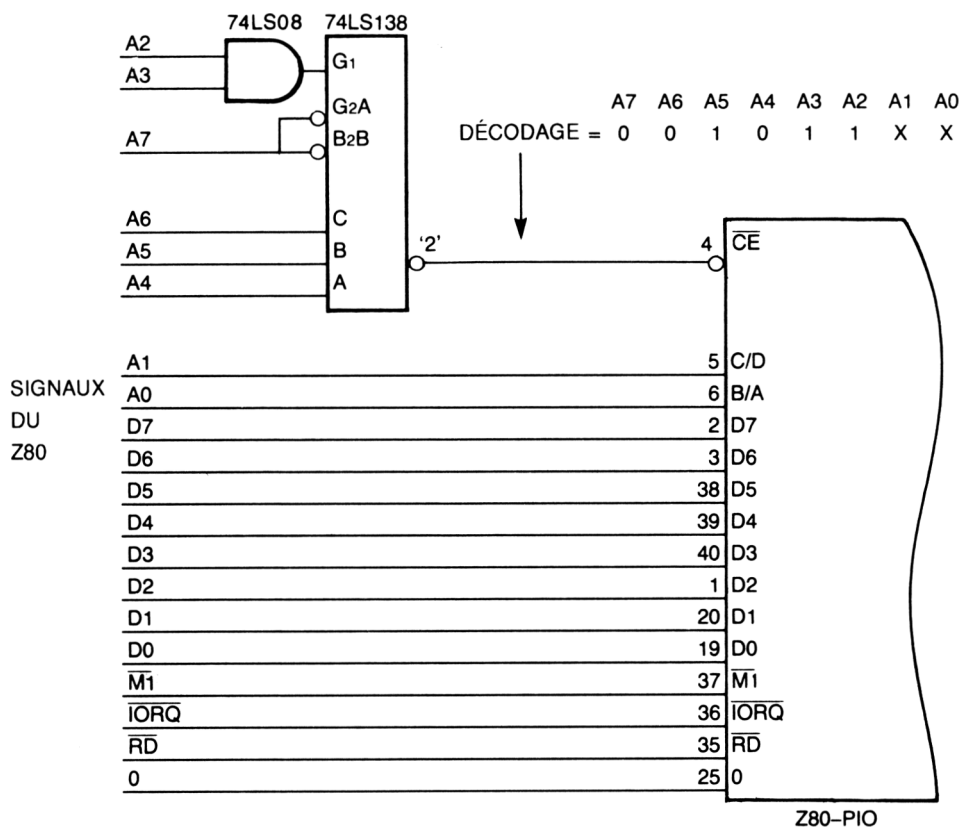


Figure 8.3 : *Schéma montrant les connexions physiques entre Z80 et PIO du Z80.*

B0-B7 Ce sont les lignes d'entrée-sortie qui relient le port B au périphérique.

$\overline{\text{BSTB}}$ (*port B STroBe pulse input*, entrée d'impulsions d'échantillonnage du port B) La fonction de cette ligne est la même que pour le port $\overline{\text{ASTB}}$ à l'exception du point suivant :

- Quand le port A est programmé en mode bidirectionnel, le signal va échantillonner les données provenant du périphérique dans l'entrée du registre A. Ceci implique que le port B n'a pas la capacité bidirectionnelle.

BRDY (*register B ReaDY*, registre B prêt) C'est une sortie active au 1 logique. La définition de cette sortie est identique au signal ARDY décrit précédemment. Une exception a lieu quand le port A est programmé en mode bidirectionnel - le signal va au 1 logique quand le registre d'entrée du port A a été lu par le Z80 et est prêt à accepter de nouvelles données provenant du périphérique.

8.3 - Connexion du PIO du Z80 au Z80

Apprenons maintenant à connecter le PIO du Z80 au Z80 de façon à avoir une communication satisfaisante. La Figure 8.3 montre la connexion type du PIO au microprocesseur Z80. Examinons les points principaux de cette figure.

La ligne de sélection B/A du PIO du Z80 est connectée physiquement à la ligne de sortie d'adresse A0 du Z80. La ligne C/D est connectée à la ligne de sortie d'adresse A1 du Z80. Les lignes d'adresses étant connectées de cette façon, les définitions de port du dispositif sont :

A1	A0	PORT	SELECTION
0	0	A	DONNEES
0	1	B	DONNEES
1	0	A	COMMANDE
1	1	B	COMMANDE

La ligne d'entrée $\overline{\text{CE}}$ du PIO du Z80 résulte d'un décodage logique des lignes d'adresses A7-A2 du système à Z80. Les adresses d'entrée-sortie de la Figure 8.3 sont 2C, 2D, 2E et 2F. Dans cette figure, les lignes d'interruption du Z80 et du PIO ne sont pas connectées. Nous les connecterons lorsque nous analyserons le fonctionnement des interruptions internes ultérieurement.

8.4 - Mise à l'état initial du PIO

Quand le PIO du Z80 est mis sous tension, il subit un processus de mise à l'état initial. Ce processus est le suivant :

1. Les registres masques des deux ports sont remis à zéro pour inhiber tous les bits de données des ports. (Comme nous ne nous sommes pas encore occupés des registres masques, il suffit pour l'instant de savoir que ces registres sont mis à l'état initial.)

2. Les lignes de bus de données de ports sont mises en état de haute impédance ; les signaux de *handshaking* READY (prêt) sont inactifs. Le mode 1 (entrée seulement) des deux ports est sélectionné automatiquement.

3. Les registres d'adresses de vecteurs d'interruptions ne sont pas remis à l'état initial. Ces registres sont les vecteurs d'interruptions qui vont être placés dans le bus de données quand le PIO recevra un accusé de réception d'interruption du Z80.

4. Les bascules d'autorisation d'interruption des deux ports sont remises à l'état initial. Ceci interdit au PIO toute réponse à une interruption jusqu'à ce qu'il soit programmé pour répondre.

5. Les registres de sortie des deux ports sont remis à l'état initial.

Ce mode de remise à zéro du PIO a lieu à la mise sous tension. On peut appliquer un circuit extérieur de remise à l'état initial en utilisant le schéma de la Figure 8.4. Comme on le voit, $\overline{M1}$ est activé sans que $\overline{RD}$ ou $\overline{IORQ}$ le soient. Cette condition ne se produira jamais avec le microprocesseur Z80 pendant l'exécution normale d'un programme.

8.5 - Programmation du PIO en mode 0 (port de sortie)

Examinons maintenant l'utilisation du PIO en mode 0. Ce mode permet d'employer les ports en sorties seulement. On peut programmer les deux ports indépendamment. Ici, nous allons programmer le port A seulement.

Nous allons commencer par supposer que le boîtier vient d'être mis à l'état initial. Le premier registre à programmer va être le registre de commande afin d'établir le mode. Les lignes d'adresses seront A0 = 0 pour le port A et C/D = 1 pour la commande. L'adresse de port 2E sera l'adresse du registre de commande du port A. Les données écrites dans le registre

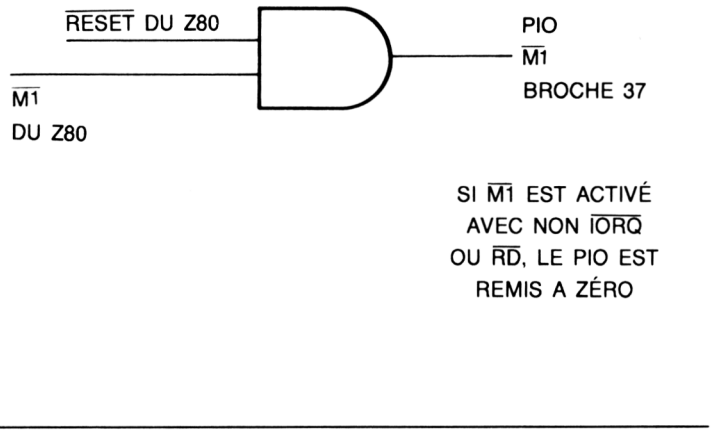


Figure 8.4 : *Circuit nécessaire pour appliquer un signal de remise à l'état initial au PIO Z80. Le PIO ne possède pas de broche distincte pour cette fonction.*

sont montrées en Figure 8.5. On peut voir dans cette figure que les quatre bits de poids faible sont mis au 1 logique, ce qui indique que le mode du port est en train d'être établi. Les bits D7 et D6 déterminent le mode. On peut donc avoir les mots suivants pour les modes : 0F (mode 0), 4F (mode 1), 8F (bidirectionnel), CF (mode de commande). Si les bits D3-D0 sont au 1 logique, les bits D4 et D5 sont ignorés.

C'est tout ce qui est nécessaire pour employer le PIO comme port de sortie. La Figure 8.6 montre un programme de Z80 qui valide le port A en sortie, puis écrit un compte binaire dans le port de sortie.

8.6 - Mode de programmation

Dans ce mode de programmation, le PIO va être mis en port d'entrée. Le mot de commande qui valide le PIO dans ce mode est 4F. Ce mot pourrait être écrit dans le port 2F ou 2E. Nous allons programmer le port B en port d'entrée en utilisant le port 2F. Dans ce mode, les données provenant du périphérique seront envoyées dans les lignes de port B0-B7, comme le montre la Figure 8.7.

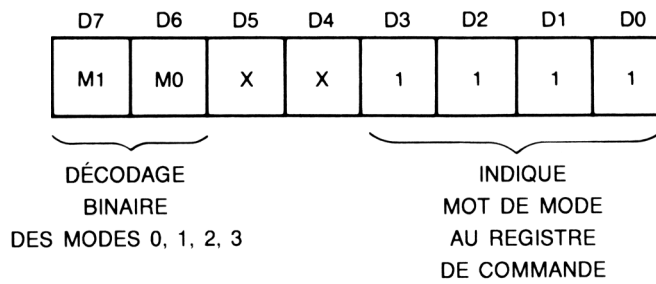


Figure 8.5 : *Les quatre bits de poids faible du mot de commande sont mis au 1 logique. Ceci indique que le mode est instauré pour le port.*

```
;
;
1800    3E0F            LD A,0FH
1802    D32E            OUT (2EH),A         ;SORTIR LE MOT DE COMMANDE
;
;   PORT A = SORTIE, B/A = 0, C/D = 1
;
;   MAINTENANT ECRIRE 53H DANS LE PORT DE SORTIE A
;
1804    3E53            LD A,53H
1806    D32C            OUT (2CH),A         ;SORTIR LES DONNEES
;
;   PROGRAMMER LE PORT B EN PORT DE SORTIE
;
1808    3E0F            LD A,0FH
180A    D32F            OUT (2FH),A         ;SORTIR LE MOT DE COMMANDE
;
;  PORT B = SORTIE, B/A = 1, C/D = 1
;
;   LE PORT B PEUT ALORS ETRE EMPLOYE EN PORT GENERAL
;   DE SORTIE EN ECRIVANT LES DONNEES DE SORTIE
;   EN ADRESSE DE PORT 2DH.
;
```

Figure 8.6 : *Programme Z80 qui valide en sortie le port A du PIO du Z80, puis écrit des octets dans le port.*

Si le port n'est pas employé en mode *handshaking,* $\overline{\text{BSTB}}$ doit être relié au 0 logique (Voir Figure 8.8). Ce type de fonctionnement en entrée lit simplement les données dans les lignes d'entrée B0-B7. La Figure 8.9 montre un programme Z80 qui initialise le PIO en mettant le port A en sortie et le port B en entrée. Ce programme lit les données dans le port B et les écrit dans le port A.

La seconde façon d'employer le mode 1 s'effectue par le mode *handshaking.* Dans ce mode, on utilise les lignes $\overline{\text{ASTB}}$, $\overline{\text{BSTB}}$ et ARDY, BRDY. La ligne $\overline{\text{BSTB}}$ échantillonne les données dans le registre d'entrée et la ligne BRDY indique que le registre d'entrée est vide, c'est-à-dire qu'il a été lu par le Z80. La Figure 8.10 donne un schéma montrant comment les signaux sont utilisés en mode *handshaking.*

Quand le périphérique écrit des données dans le PIO, la ligne d'interruption devient active. Dans la routine de service d'interruption, le Z80 lit les données provenant du port d'entrée, mettant ainsi la ligne BRDY au 1 logique et indiquant au périphérique que d'autres données peuvent être envoyées au PIO.

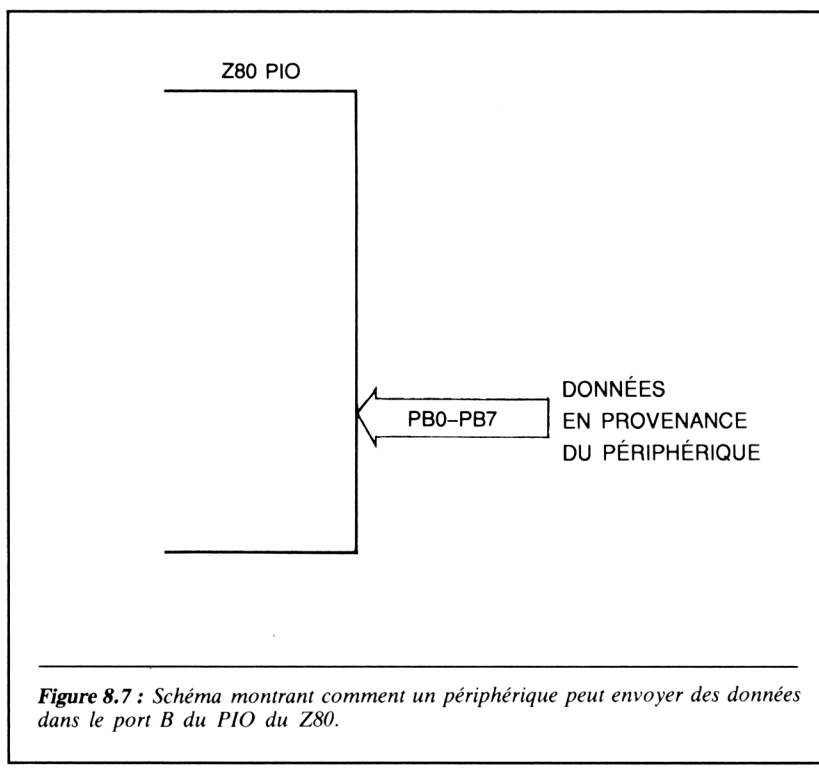


Figure 8.7 : *Schéma montrant comment un périphérique peut envoyer des données dans le port B du PIO du Z80.*

Si on utilise ce dernier dans ce mode, il faut effectuer une lecture fictive dans le port d'entrée afin de mettre la ligne BRDY au 1 logique. Il faut le faire car, à la mise à l'état initial, la ligne BRDY est mise au 0 logique.

On peut aussi employer le mode 0 en mode *handshaking*. Dans ce mode, la ligne RDY va au 1 logique quand le Z80 a écrit des données dans le port de sortie. Quand le périphérique lit ces données, il accuse réception de l'opération de lecture en activant l'entrée STB. A ce moment-là, la ligne RDY va au 0 logique et la ligne de sortie $\overline{INT}$ devient activée. La Figure 8.11 présente un schéma expliquant l'emploi de ce mode.

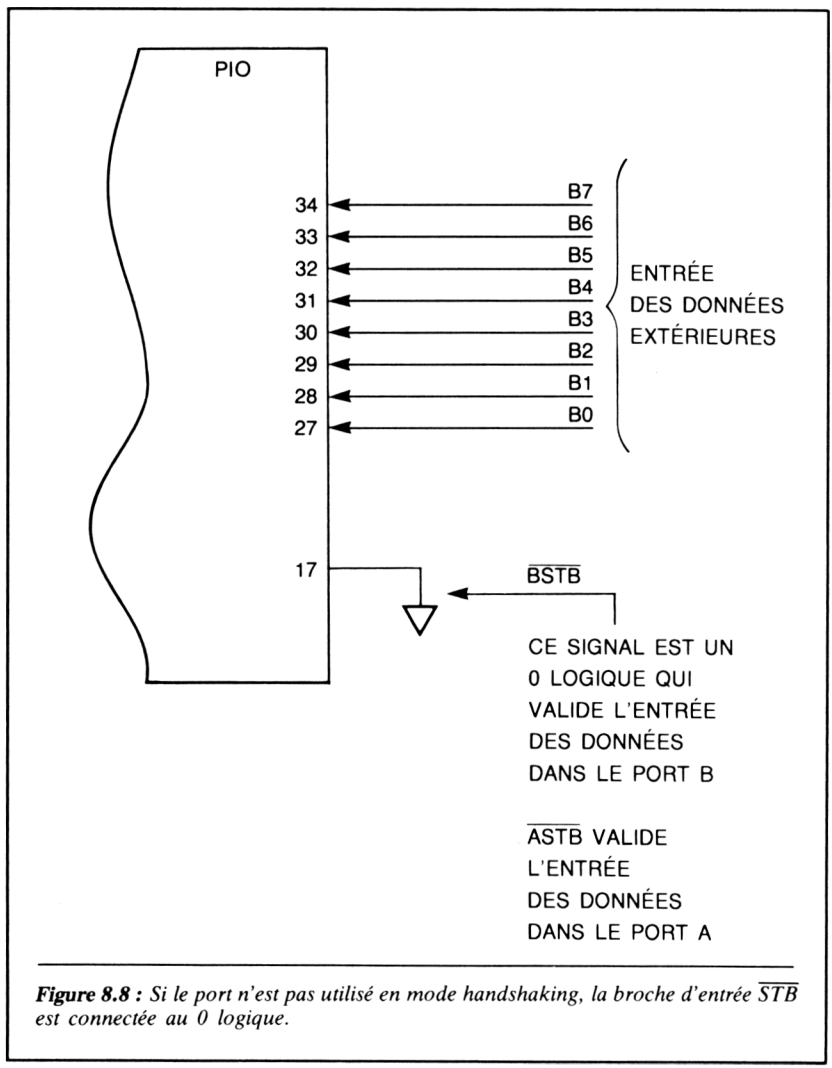


Figure 8.8 : *Si le port n'est pas utilisé en mode handshaking, la broche d'entrée $\overline{STB}$ est connectée au 0 logique.*

```
1800   3E0F            LD A,0FH          ;A REG = MOT DE COMMANDE
1802   D32E            OUT (2EH),A       ;ECRIRE MOT DE COMMANDE DANS PIO
;
;  LE PORT A EST MAINTENANT UN PORT DE SORTIE
;
1804   3E4F            LD A,4FH          ;A REG = MOT DE COMMANDE
1806   D32F            OUT (2FH),A       ;ECRIRE MOT DE COMMANDE DANS PIO
;
;  LE PORT B EST MAINTENANT UN PORT D'ENTREE
;
1808   DB2D            IN A,(2DH)        ;LIRE DONNEES DANS PORT B
180A   D32C            OUT (2CH),A       ;SORTIES DE DONNEES DANS PORT A
;
```

Figure 8.9 : *Programme Z80 qui va initialiser le PIO en mettant le port A en sortie et le port B en entrée, permettant ainsi au programme de lire les données dans le port A et d'en écrire dans le port B.*

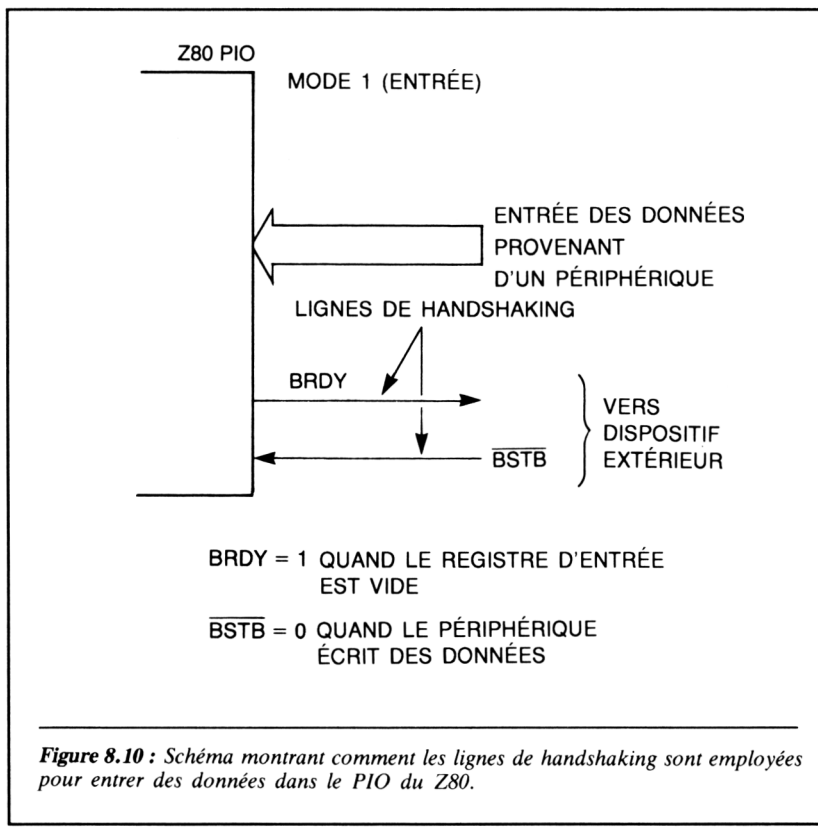


Figure 8.10 : *Schéma montrant comment les lignes de handshaking sont employées pour entrer des données dans le PIO du Z80.*

8.7 - Etablissement du mot de commande d'interruption

Dans les parties précédentes, nous avons programmé le PIO en mode 0 et en mode 1. Pour ces deux modes, nous avons expliqué les interruptions, mais nous n'avons pas montré comment les utiliser. Voyons maintenant comment on peut programmer et utiliser les interruptions dans ces deux modes.

Pour établir le mot de commande d'interruption, il faut écrire dans le registre de commande d'un port particulier. Les quatre bits de poids faible, D3-D0, sont mis à 0111. Ceci indique que les quatre bits de poids fort vont définir le mot de commande d'interruption. La Figure 8.12 montre la signification de chaque bit du mot de commande d'interruption.

Si le bit D7 du mot de commande d'interruption est mis au 1 logique, la bascule d'autorisation d'interruption du PIO est mise à l'état initial et le dispositif peut produire des interruptions. Si le bit est effacé ou mis au 0 logique, les interruptions sont interdites. Si une condition d'interruption se présente alors que les interruptions sont interdites, le PIO va produire une

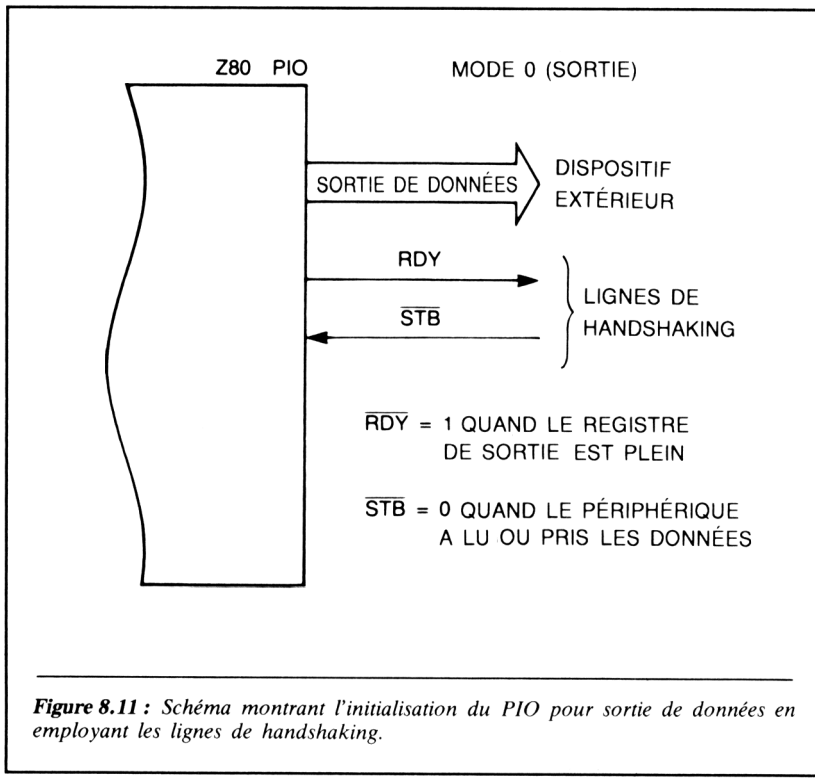


Figure 8.11 : *Schéma montrant l'initialisation du PIO pour sortie de données en employant les lignes de handshaking.*

demande d'interruption, quand les interruptions seront à nouveau autorisées. Si le bit D4 du mot de commande d'interruption est mis au 1 logique, toute interruption en attente va être remise à zéro dans le PIO.

Les bits D6, D5 et D4 ne servent qu'en mode 3. Nous y reviendrons plus loin dans ce chapitre. Pour les interruptions en modes 0 et 1, il suffit d'autoriser ou d'interdire les circuits d'interruptions du PIO.

La bascule d'autorisation d'interruption peut être validée ou inhibée en envoyant les mots suivants dans le registre de commande de port :

```
AUTORISATION D'INTERRUPTIONS                87H
INHIBITION D'INTERRUPTIONS                  07H
```

On peut aussi autoriser ou inhiber les interruptions sans modifier le reste du mot de commande d'interruption en écrivant 83H pour autoriser ou 03H pour inhiber.

Si les interruptions sont autorisées, le programme doit aussi spécifier le vecteur d'interruption qui doit être chargé dans le bus de données et employé par le Z80 en mode d'interruption 2. Le vecteur d'interruption peut être chargé dans le PIO en écrivant dans le registre de commande du

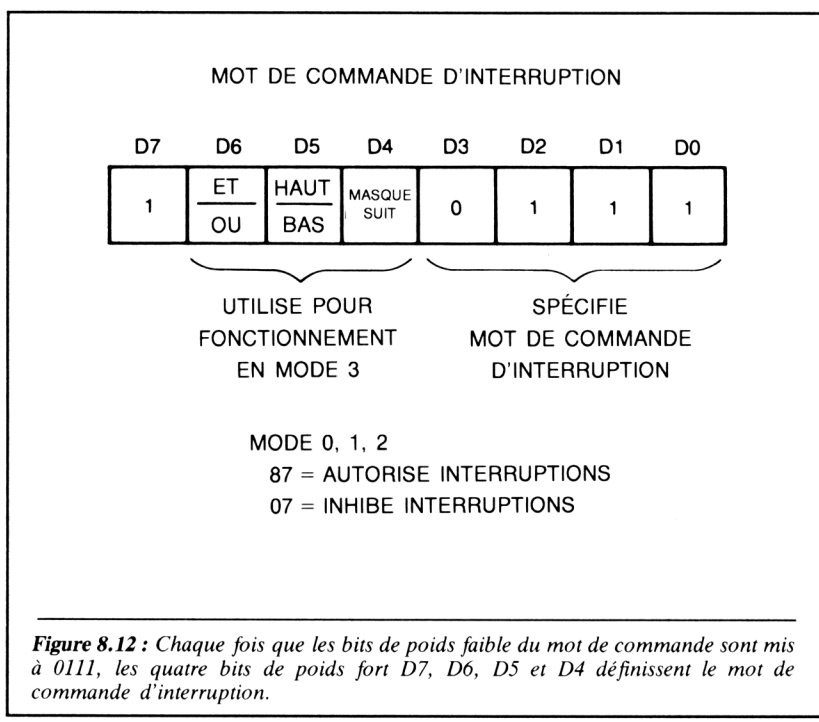


Figure 8.12 : *Chaque fois que les bits de poids faible du mot de commande sont mis à 0111, les quatre bits de poids fort D7, D6, D5 et D4 définissent le mot de commande d'interruption.*

port choisi la configuration du mot de la Figure 8.13. La Figure 8.14 montre un programme partiel de Z80 qui autorise les interruptions du port A, et établit le port en sortie. Le vecteur d'interruption écrit dans le PIO est égal à 76H.

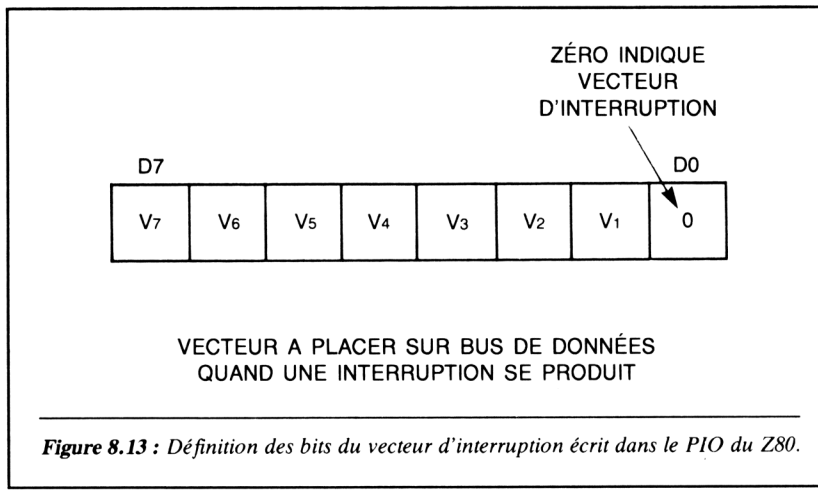


Figure 8.13 : *Définition des bits du vecteur d'interruption écrit dans le PIO du Z80.*

```
;
1800   3E76            LD A,76H        ;VECTEUR D'INTERRUPTION = 76H
1802   D32E            OUT (2EH),A     ;VECTEUR DE SORTIE VERS PIO
;
1804   3E0F            LD A,0FH        ;MOT DE COMMANDE DU PORT A
1806   D32E            OUT (2EH),A     ;PORT A EST EN SORTIE
;
;  MAINTENANT ECRIRE LE MOT DE COMMANDE D'INTERRUPTION
;
1808   3E87            LD A,87H
180A   D32E            OUT (2EH),A     ;UN MOYEN D'AUTORISER INTERRUPTION
;
; UN AUTRE MOYEN D'AUTORISER INTERRUPTION EST LE
; SUIVANT
180C   3E83            LD A,83H
180E   D32E            OUT (2EH),A     ;CECI AUTORISE INTERRUPTIONS
;
;  IL N'EST PAS NECESSAIRE D'EXECUTER LES DEUX TYPES
;  D'AUTORISATION D'INTERRUPTIONS
;
```

Figure 8.14 : *Programme Z80 pour écrire le vecteur d'interruption 76H dans le PIO du Z80 et pour valider l'interruption.*

8.8 - Examen des temps en modes 0 et 1

Maintenant que nous avons examiné des exemples montrant comment fonctionne le PIO dans les modes 0 et 1, le diagramme des temps de la Figure 8.15 devrait être plus facile à comprendre.

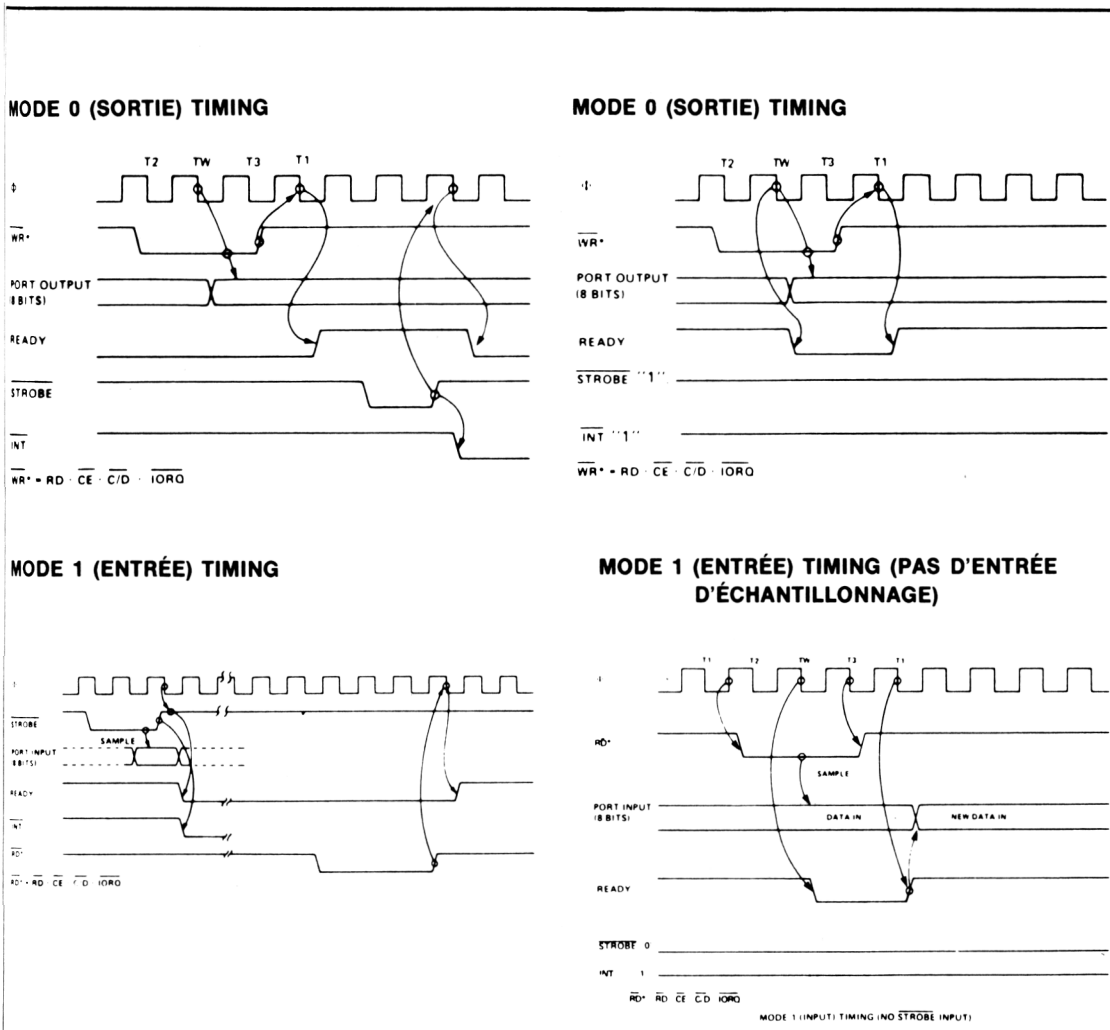


Figure 8.15 : *Diagramme des temps de fonctionnement en mode 0 et en mode 1 du PIO du Z80.*

8.9 - Utilisation du PIO en mode bidirectionnel (mode 2)

Le mode de fonctionnement bidirectionnel est une combinaison des modes 0 et 1. Il utilise les quatre lignes de *handshaking* du PIO. A cause de cela, le mode d'entrée-sortie n'existe que sur le port A. Le port B doit être mis en mode de commande. Le vecteur d'interruption de sortie doit être programmé dans le port A, tandis que le vecteur d'interruption d'entrée doit être programmé dans le port B.

La Figure 8.16 montre le déroulement d'un transfert de données type avec PIO en mode bidirectionnel. Examinons maintenant certains points du diagramme des temps. Les lignes de *handshaking* du port A servent à la commande de sortie et les lignes du port B servent à la commande d'entrée. Chaque ligne de *handshaking* fonctionne comme si le port sélectionné était programmé en mode 0 ou en mode 1. Commençons par le transfert de données du PIO vers le périphérique.

1. Le Z80 programme d'abord le PIO pour utilisation en mode bidirectionnel.

2. Le Z80 écrit ensuite un mot de données dans le registre de sortie du port A. A ce moment-là, la ligne de sortie ARDY va au 1 logique, indiquant électriquement au périphérique que des données sont prêtes à être extraites du PIO.

3. Le périphérique active ensuite l'entrée $\overline{\text{ASTB}}$ du PIO. Ceci met les données de sortie dans les lignes de données A0-A7. Quand le périphérique a lu les données, la ligne $\overline{\text{ASTB}}$ est inactivée au 1 logique.

4. La ligne de sortie $\overline{\text{INT}}$ devient ensuite active, signalant ainsi au Z80 que les données de sortie ont été prises.

5. Supposons maintenant que le Z80 reçoive des données d'un périphérique. La ligne $\overline{\text{ASTB}}$ est au 1 logique. Le périphérique met les données dans les lignes A0-A7, car il active le signal d'entrée $\overline{\text{BSTB}}$. Quand le signal $\overline{\text{BSTB}}$ passe du 0 au 1 logique (comme le montre la Figure 8.16), la ligne de sortie $\overline{\text{INT}}$ devient activée. Ceci indique au Z80 que le périphérique a envoyé des données au PIO et attend maintenant que le Z80 les lise.

6. Quand $\overline{\text{INT}}$ est activé, la ligne de sortie BRDY passe au 0 logique. Ceci indique au périphérique que le Z80 n'a pas lu les données et qu'il ne faut plus envoyer de données jusqu'à ce que BRDY aille à nouveau au 1 logique. Cette ligne ira au 1 logique quand le Z80 lira les données du port A.

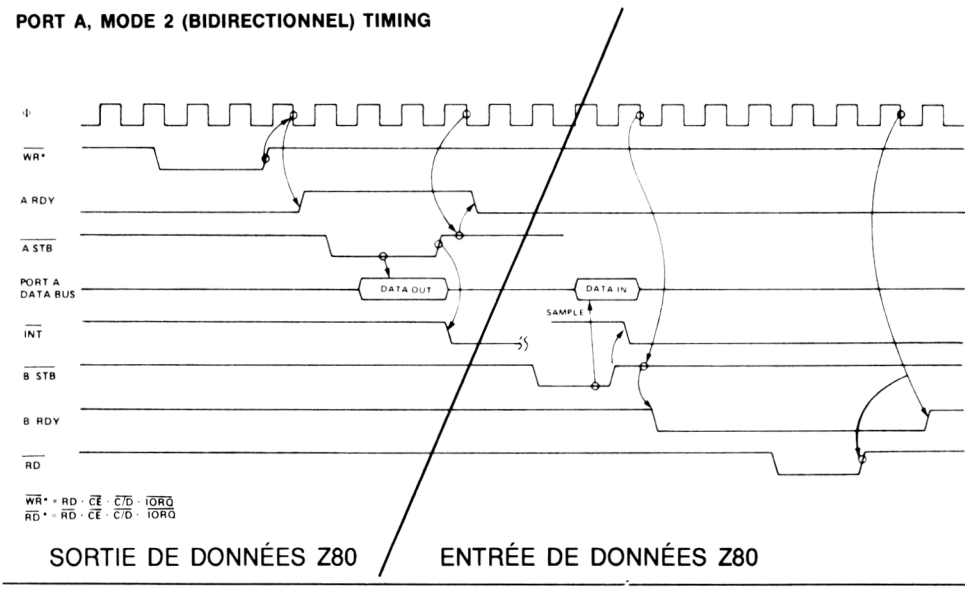


Figure 8.16 : *Diagramme des temps d'un transfert de données type employant le PIO en mode bidirectionnel.*

De l'analyse précédente, on peut déduire qu'une interruption est demandée quand le périphérique a pris les données de sortie (port $\overline{\text{ASTB}}$) ou bien quand il a écrit des données dans le PIO (port $\overline{\text{BSTB}}$). Dans chaque cas, le Z80 a besoin de savoir lequel de ces deux fonctionnements a eu lieu. Ceci est effectué en chargeant un vecteur d'interruption différent dans les ports A et B.

Quand le périphérique écrit des données dans le PIO, le vecteur d'interruption B est mis dans le bus de données du système. Quand le PIO écrit des données dans le périphérique et que les données ont été acceptées, le vecteur d'interruption du port A est mis dans le bus de données.

Dans ce mode de fonctionnement, il est possible que le port B provoque une interruption due à un fonctionnement en mode 3, plutôt que le fonctionnement de la ligne de commande que nous désirons. Pour éviter cela, il faut s'assurer que le port B est empêché de causer des interruptions dues à ce type de fonctionnement. Bien que nous n'ayons pas encore analysé les interruptions en mode 3, nous mentionnons ici ce fait, parce que cela pourrait être une source de difficultés lors d'utilisation du PIO en mode 2.

```
;
2700            OUTDAT  EQU 2700H       ;MEM LOC POUR DONNEES DE SORTIE
2701            INDAT   EQU OUTDAT+1    ;MEM LOC POUR DONNEES D'ENTREE
;
;
;
;   CE PROGRAMME VA ETABLIR LE PORT A EN PORT D'E/S
;
1800    3E8F            LD A,8FH        ;C'EST LE MOT DE MODE
1802    D32E            OUT (2EH),A     ;ECRIRE MOT DE MODE DANS PA
1804    3E00            LD A,00H        ;VECTEUR D'INTERRUPT. POUR PA
1806    D32E            OUT (2EH),A     ;ECRIRE VECTEUR DANS PA
;
;
1808    3ECF            LD A,0CFH       ;MOT DE MODE POUR PORT B
180A    D32F            OUT (2FH),A     ;ECRIRE MOT DE MODE DANS PORT B
;
;   MOT DE MODE ETABLIT PORT B EN MODE 3
;   LES DEFINITIONS DE BITS POUR LE PORT B SUIVENT
;
180C    3EFF            LD A,0FFH       ;TOUS LES BITS SONT DES ENTREES
180E    D32F            OUT (2FH),A     ;ECRIRE DANS PORT B
1810    3E17            LD A,17H        ;MOT DE COMMANDE D'INT. DU PORT B
1812    D32E            OUT (2EH),A     ;INHIBER INT, MASQUE SUIT
1814    3EFF            LD A,0FFH       ;TOUS LES BITS INHIBES
1816    D32F            OUT (2FH),A     ;ECRIRE MASQUE DANS PORT B
;
1818    3E02            LD A,02H        ;VECTEUR INTERRUPT. PORT B
181A    D32F            OUT (2FH),A     ;ECRIRE DANS PORT B
;
181C    3E3A            LD A,3AH        ;ETABLIR OCTET HAUT DE TABLE
181E    ED47            LD I,A          ;DES VECTEURS
;
;   TABLE DE VECTEUR EN 3A00 POUR LE PORT A
;   ET EN 3A02 POUR LE PORT B
;   ADRESSE DES ROUTINES DE SERVICE EN FC81
;   POUR PORT A, ET FD00 POUR PORT B
;
;   L'INTERRUPTION DU PORT A EST POUR EMETTRE DES DONNEES
;   L'INTERRUPTION DU PORT B EST POUR ENTRER DES DONNEES
;
1820    ED5E            IM 2            ;ETABLIR MODE 2 D'INTERRUPTION
1822    3E83            LD A,83H        ;VALIDER INT. SUR PIO
1824    D32E            OUT (2EH),A     ;VALIDER PORT A
1826    D32F            OUT (2FH),A     ;VALIDER PORT B
1828    FB              EI              ;VALIDER INT DU Z80
;
;
1829    C32918  BOUCLE  JP BOUCLE       ;ATTENDRE INTERRUPTIONS
```

Figure 8.17 : *Programme Z80 pour établir le PIO en mode 2. Les routines de service d'interruptions sont employées pour entrer et émettre des données avec le PIO.*

```
;
;
; LA SUITE SERT A INITIALISER LES VECTEURS D'INTERRUPTION
; ET LES ROUTINES D'INTERRUPTIONS EFFECTIVES
;
3A00                    CODE 3A00H
3A00   81FC             DEFW  0FC81H     ;VECTEUR D'INTERRUPT. DU PORT A
3A02   00FD             DEFW  0FD00H     ;VECTEUR D'INTERRUPT. DU PORT B
;
;  VOICI LES ROUTINES REELLES
;
FC81                    CODE 0FC81H
;
FC81   3A0027           LD A,(OUTDAT)    ;CHARGER ACC. AVEC DONNEES DE SORTIE
FC84   D32C             OUT (2CH),A      ;ECRIRE MOT DANS PORT A
FC86   FB               EI               ;VALIDER INTERRUPTIONS
FC87   ED4D             RETI             ;RETOUR D'INTERRUPTION
;
;
FD00                    CODE 0FD00H
;
FD00   D82D             IN A,(2DH)       ;LIRE DONNEES DE PORT B
FD02   320127           LD (INDAT),A     ;STOCKER DONNEES EN MEMOIRE
FD05   FB               EI               ;VALIDER INTERRUPTIONS
FD06   ED4D             RETI             ;RETOUR D'INTERRUPTIONS
;
                        END
```

Figure 8.17 : *Programme Z80 (suite).*

La Figure 8.17 montre un programme Z80 destiné à mettre le PIO en mode 2. Des routines de service d'interruptions sont utilisées pour entrer et sortir des données. On suppose que le périphérique va activer $\overline{\text{ASTB}}$ et $\overline{\text{BSTB}}$ au moment voulu.

8.10 - Utilisation du PIO en mode 3

Le fonctionnement du PIO en mode 3 est prévu en configuration sans *handshaking*. Les huit bits de chaque port peuvent se voir attribuer individuellement une fonction d'entrée ou de sortie dans n'importe quel ordre. Les bits B0, B4 et B5 peuvent, par exemple, être des entrées et tous les autres bits de données du port B des sorties. Voici comment cela est effectué.

Le programmeur écrit d'abord le mot de mode dans le registre de commande du port choisi. Pour sélectionner le mode 3, les données écrites dans le registre de commande sont 0CFH. Le mot suivant immédiatement

ce mot de sélection de mode va définir logiquement l'emploi des huit bits de port. Un 1 logique dans la position du bit va mettre la ligne de donnée en entrée. Un 0 logique va la mettre en sortie. Les instructions suivantes vont mettre le PIO en mode 3 et définir les bits B0, B3 et B6 en entrées. Les lignes B1, B2, B4, B5 et B7 seront des sorties.

```
LD A,0CFH       CHARGER CF DANS LE REGISTRE A
OUT (2FH),A     SELECTIONNER LE MODE 3 POUR LE REGISTRE B
LD A,49H        BITS D0, D3 et D6 = 1
OUT (2FH),A     METTRE B0, B3 et B6 EN ENTREES
```

Le PIO est maintenant initialisé avec les définitions de bits d'entrée-sortie appropriées. Un programmeur peut, à présent, envoyer des données dans le port de sortie et en lire dans le port d'entrée. Quand on lit des données dans le port d'entrée, on lit la valeur logique des bits d'entrée programmés. Les bits programmés en sortie renvoient la dernière valeur logique envoyée au registre pendant une instruction OUT. Autrement dit, tous les bits vont être lus pendant l'exécution de l'instruction d'entrée. Cependant, seuls les bits programmés en entrée représenteront des données d'entrée valables.

Une autre caractéristique du fonctionnement en mode 3, c'est la capacité d'interruption. Le PIO peut obliger la production d'une demande d'interruption chaque fois qu'il y a dans les lignes du port d'entrée une combinaison logique particulière. Il y a deux parties (de programmation) pour établir ceci dans le PIO. L'une sert à spécifier le mot de commande d'interruption. Souvenez-vous que dans les modes 0, 1 et 2, nous avons spécifié le mot de commande d'interruption et avons ignoré les bits D6, D5 et D4. Nous allons maintenant utiliser ces bits.

Le bit D6 du mot de commande d'interruption définit comment doit être formée logiquement la configuration d'entrée que nous recherchons dans le port d'entrée. Les deux choix consistent à effectuer une opération ET et une opération OU. Si on choisit la fonction ET, tous les bits sélectionnés doivent avoir l'état actif avant la production d'une interruption. Si on choisit la fonction OU, une interruption est produite si l'un quelconque des bits sélectionnés va à l'état actif.

Il ne faut pas considérer ces fonctions ET et OU comme de simples fonctions booléennes. On doit les considérer comme effectuant des opérations ET et OU avec les états actifs.

L'état actif peut être un 1 logique ou un 0 logique. Cependant, on doit considérer toutes les entrées dans le même état actif. Autrement dit, toutes les entrées sélectionnées doivent être considérées en 1 logique ou en 0 logique.

Le bit D5 détermine l'état actif des lignes de données du port choisi. Quand D5 est au 1 logique, on surveille l'apparition d'un 1 logique dans les

lignes de données de port. Quand D5 est au 0 logique, on surveille l'apparition d'un 0 logique dans les lignes de données de port.

Après sélection du ET et du OU d'entrée et spécification de l'état actif, il est nécessaire de définir lequel des huit bits doit être examiné. Si le bit D4 du mot de commande d'interruption est au 1 logique, la prochaine donnée envoyée au registre de commande sera le mot de masque. La Figure 8.18 montre le mot de bit de masque. Si le programmeur veut examiner un bit particulier du bus de données de port, le bit de masque est mis au 1 logique. En d'autres termes, tout bit que l'on ne souhaite pas examiner logiquement est masqué.

Pour expliquer complètement comment ce mode peut fonctionner avec le PIO du Z80, étudions un exemple :

1. On met d'abord le port B en mode 3.

2. On programme ensuite en entrées les lignes de données B2, B3 et B4 du port B.

3. On programme ensuite en sorties les lignes de données B0, B1, B5, B6 et B7 du port B.

4. On surveille ensuite une interruption dans les lignes de données B3 et B4 du port B. On masque toutes les autres lignes du port B.

5. L'état actif de B3 et B4 sera un 0 logique.

6. Un mot se produira quand les deux bits seront au 0 logique actif. C'est la fonction ET des états actifs.

7. Le vecteur d'interruption sera 08.

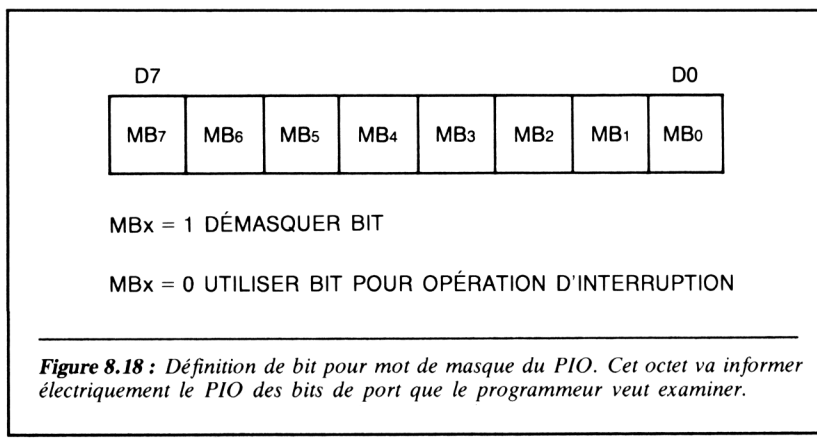


Figure 8.18 : *Définition de bit pour mot de masque du PIO. Cet octet va informer électriquement le PIO des bits de port que le programmeur veut examiner.*

On suppose, dans cet exemple, que le PIO est situé dans l'espace d'entrée-sortie comme défini précédemment, c'est-à-dire avec les ports d'entrée-sortie 2C, 2D, 2E et 2F. La Figure 8.19 montre le schéma du périphérique connecté au PIO.

Voici le programme Z80 permettant cette configuration :

```
LD A,11001111B
OUT (2FH),A        ETABLIR MODE 3 DANS LE PIO POUR LE
                   PORT B
LD A,00011100B
OUT (2FH),A        METTRE BITS B2, B3, B4 EN ENTREES
LD A,00001000B
OUT (2FH),A        VECTEUR D'INT. = 08
LD A,11010111B
OUT (2FH),A        AUTORISER INT. BAS, MASQUE SUIT
LD A,11100111B
OUT (2FH),A        LES BITS B3, B4 NE SONT PAS MASQUES
```

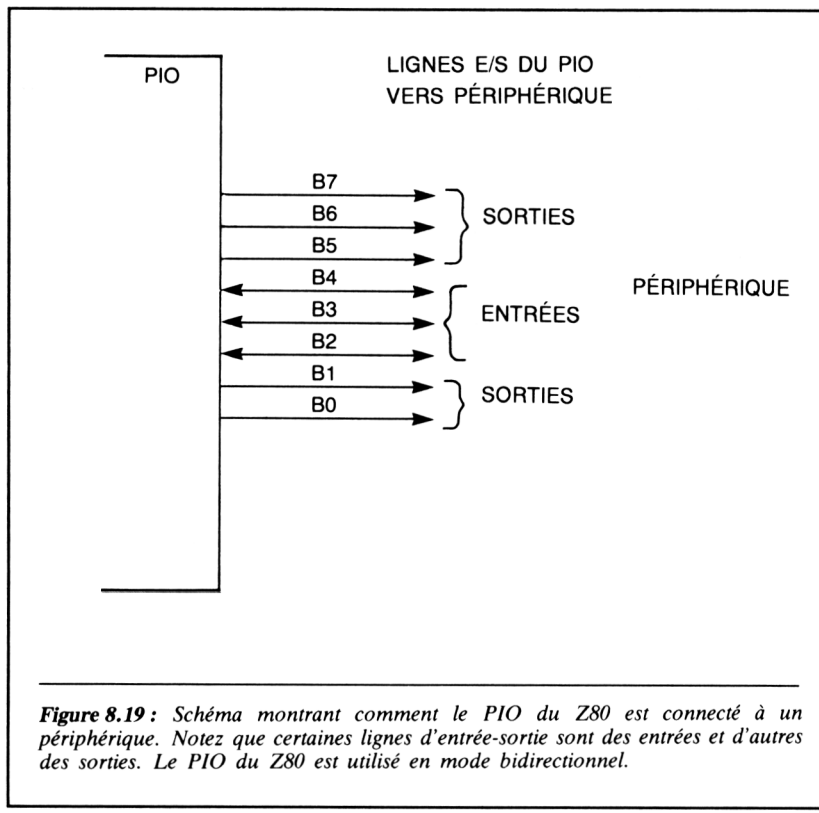


Figure 8.19 : *Schéma montrant comment le PIO du Z80 est connecté à un périphérique. Notez que certaines lignes d'entrée-sortie sont des entrées et d'autres des sorties. Le PIO du Z80 est utilisé en mode bidirectionnel.*

Le programme précédent configurera le PIO comme décrit dans la définition précédente. Une autre partie du programme qui doit être écrite est la routine de service d'interruption. (On peut se reporter au Chapitre 4, et à l'analyse des interruptions du Z80.)

Une caractéristique séduisante du PIO est qu'il surveille le bus de données du Z80 ; quand une instruction RETI est cherchée en mémoire, il supprime automatiquement l'interruption. Rappelez-vous que, puisque c'est le travail normal du programmeur de retirer explicitement la demande d'interruption pendant la routine de service d'interruption, cette caractéristique du PIO soulage le programmeur qui n'a pas à effectuer cette tâche.

8.11 - Autorisation et inhibition d'interruption

Quand le PIO est mis en configuration initiale, il se peut que survienne une demande d'interruption extérieure indésirable. Ceci peut créer une difficulté au système étant donné que la condition d'interruption n'existe pas réellement. Pour éliminer la possibilité que cela survienne, il faut inhiber les entrées d'interruptions provenant du Z80 lorsque l'on configure le PIO. Une autre solution consiste à inhiber simplement les interruptions du Z80 ; le programme inhibera ensuite les interruptions provenant du PIO. On peut accomplir cette tâche par les instructions suivantes :

```
LD A,03H
DI              INHIBER INTERRUPTIONS DU Z80
OUT (2EH),A     INHIBER INTERRUPTIONS SUR PORT A DU PIO
OUT (2FH),A     INHIBER INTERRUPTIONS SUR PORT B DU PIO
EI              VALIDER INTERRUPTIONS SUR Z80
```

Le programme précédent permet d'inhiber le PIO : le Z80 peut avoir ensuite son système propre d'interruptions revalidé.

8.12 - Priorité d'interruption du PIO

Nous allons maintenant examiner les trois lignes de commande d'interruptions du PIO : $\overline{\text{INT}}$, IEO, IEI. La ligne $\overline{\text{INT}}$ est une sortie ; elle est reliée à la ligne d'entrée $\overline{\text{INT}}$ du Z80. IEI est la ligne d'entrée d'autorisation d'interruption. Si cette ligne est au 1 logique, le PIO est électriquement

capable de causer une interruption. Si IEI est au 0 logique, la sortie d'interruption $\overline{INT}$ est inhibée.

Quand le dispositif sélectionné du PIO demande activement l'interruption, la sortie IEO va au 0 logique, indiquant ainsi que ce dispositif est actuellement en service. En utilisant ces lignes (IEO et IEI), on peut chaîner jusqu'à quatre PIO en configuration de priorité d'interruption, comme on le voit sur la Figure 8.20. Dans cette figure, le dispositif ayant la priorité la plus élevée a sa ligne d'entrée IEI connectée à un 1 logique.

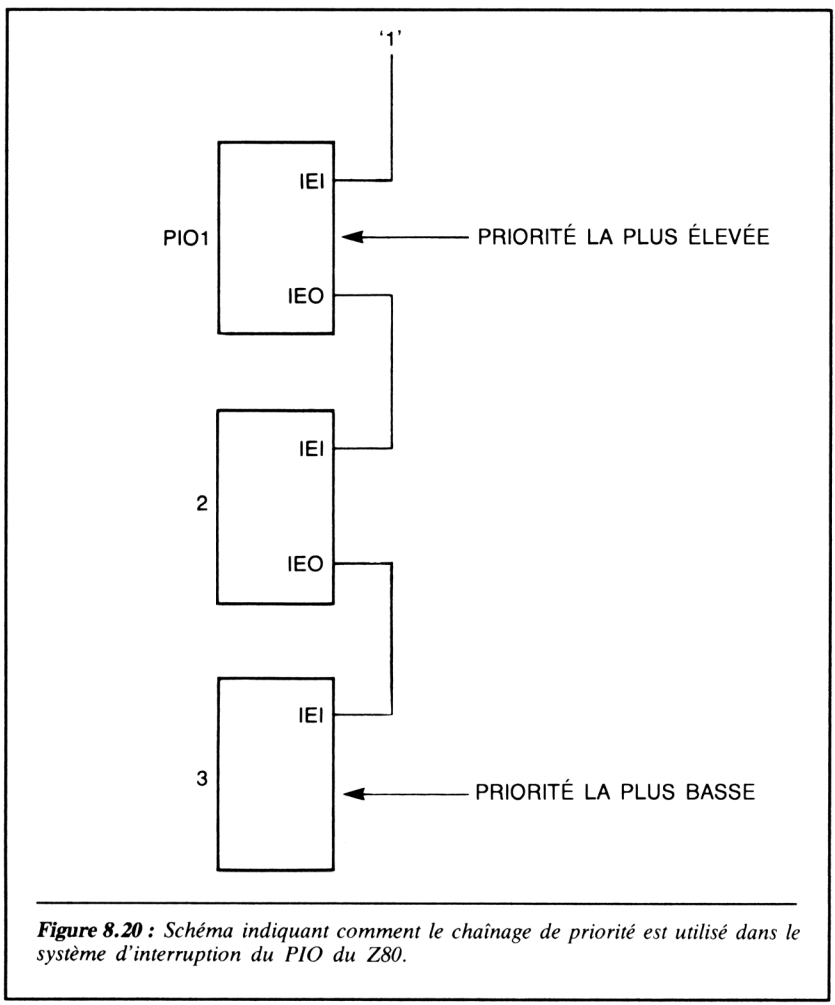


Figure 8.20 : *Schéma indiquant comment le chaînage de priorité est utilisé dans le système d'interruption du PIO du Z80.*

RESUME

Dans ce chapitre, nous avons examiné les principales caractéristiques du PIO du Z80. Nous avons commencé par examiner le schéma du PIO et continué par une analyse de chacun des modes de fonctionnement. En présentant ces modes de fonctionnement, nous avons donné des exemples de programmes Z80 permettant de mettre le PIO dans la configuration appropriée.

Nous avons montré et expliqué les circuits destinés à interfacer le PIO au Z80. Les renseignements donnés dans ce chapitre s'appliquent à presque toutes les réalisations de systèmes.

UTILISATION DU CTC DU Z80

CHAPITRE 9

INTRODUCTION

Dans ce chapitre, nous allons étudier le boîtier compteur temporisateur du Z80, appelé aussi CTC du Z80. C'est un boîtier LSI spécialement conçu pour être utilisé avec le microprocesseur Z80. Il est d'un emploi très souple. Au fur et à mesure de ce chapitre, nous allons examiner les caractéristiques spéciales qui contribuent à sa souplesse.

Nous commencerons par examiner un schéma et par passer en revue les registres et les liaisons. Vers la fin de ce chapitre, vous devriez pouvoir programmer et connecter le CTC à votre propre système à Z80 pour effectuer les fonctions désirées.

9.1 - Schéma du CTC

Ce boîtier tire son nom de deux fonctions essentielles qu'il effectue : comptage et temporisation. Il y a quatre canaux de comptage-temporisation indépendants dans un seul boîtier DIP à 24 broches.

La Figure 9.1 en montre un schéma. Examinons-le.

On peut voir dans cette figure que le CTC a quatre canaux indépendants, presque identiques, nommés Ch0, Ch1, Ch2 et Ch3. Connectées à trois canaux (0, 1 et 2), deux lignes de signaux appelées compte zéro/fin du temps (sortie) et déclenchement d'horloge (entrée) s'interfacent avec un périphérique. Le canal 3 n'a que la ligne d'entrée de déclenchement. Ceci est dû au nombre limité de broches du boîtier DIP à 24 broches.

On voit aussi, dans la Figure 9.1, la *logique de commande interne*. Ce bloc assure la synchronisation appropriée de tous les transferts de données à l'intérieur du composant.

La Figure 9.1 montre aussi un bloc logique de commande d'interruption. La structure d'interruption de ce dispositif est très puissante. Nous examinerons plus tard en détail l'utilisation de la logique d'interruption.

Le dernier bloc de la Figure 9.1 est l'*entrée-sortie du bus du CPU*. Cette logique donne l'interface entre Z80 et CTC. Il y a huit lignes de données et six lignes de commandes connectées à ce bloc. C'est par ce bloc que le Z80 communique électriquement avec le CTC, de sorte qu'il peut être programmé pour agir exactement comme le veut le programmeur.

9.2 - Examen plus détaillé du bloc de canaux

On peut voir dans la Figure 9.1 comment tous les principaux blocs du CTC réagissent entre eux. Pour programmer et utiliser le CTC, nous devons cependant y regarder de plus près. Dans ce but, examinons d'une façon plus détaillée le bloc référencé canal 0. (Notez que cet examen est valable également pour tous les autres canaux, excepté naturellement le canal 3, car il n'a pas de ligne de sortie compte zéro/fin de temps.)

La Figure 9.2 montre le schéma d'un canal. On peut voir dans cette figure qu'un seul canal comporte un registre de commande de canal, un registre de constante de temps et un décompteur. Examinons ces sous-ensembles.

Le programme écrit des informations dans le *registre de commande de canal*. Ce registre définit en effet le fonctionnement du canal.

Près du registre de commande de canal se trouve le *registre de constante de temps*. Ce registre a une étendue de 8 bits. Il contient un nombre binaire allant de 00H à FFH. Ce nombre sert à initialiser le bloc décompteur.

Ce *bloc décompteur* est un décompteur à 8 bits. Sa sortie est référencée compte zéro/fin de temps. Quand le compteur atteint le compte final ou zéro, sa sortie devient active d'une façon programmée dans le dispositif.

SCHÉMA DU CTC DU Z80

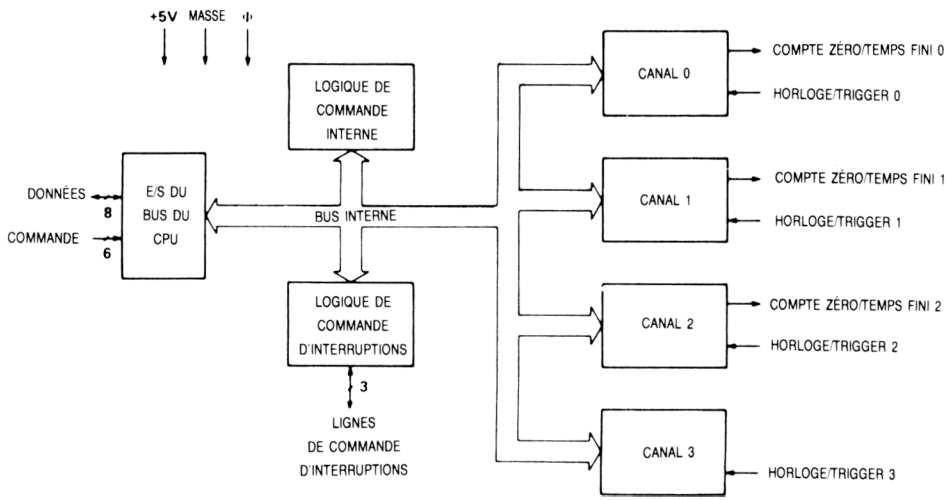


Figure 9.1 : *Schéma du CTC du Z80. Remarquez qu'il y a, dans le boîtier, quatre compteurs presque identiques.*

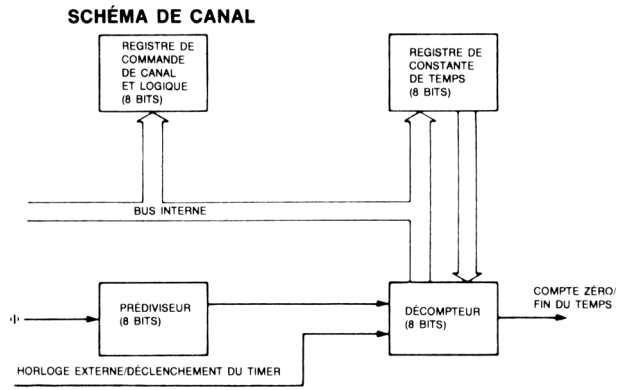


Figure 9.2 : *Schéma d'un canal de compteur isolé dans l'ensemble du dispositif. Ce schéma va servir pour les quatre canaux des compteurs internes.*

En face du bloc décompteur se trouve un bloc nommé *compteur à prédétermination* (8 bits). Ce bloc prédétermine ou divise l'entrée d'horloge du décompteur par 16 ou 256, suivant la programmation du boîtier.

Une autre entrée du décompteur est une ligne nommée *horloge externe/déclenchement de temporisateur*. Cette ligne peut être une entrée d'horloge directe du décompteur à 8 bits ou bien elle peut servir de ligne de validation à l'horloge de sortie du prédiviseur (Voir Figure 9.2). La fonction de cette ligne dépend de la façon dont le fonctionnement du CTC a été programmé.

9.3 - Brochage du CTC du Z80

Avant de commencer à programmer le CTC, assurons-nous que nous pouvons le connecter au microprocesseur Z80. Après, nous pourrons nous efforcer d'écrire les données adéquates dans le CTC pour un fonctionnement approprié. La Figure 9.3 montre le brochage et les noms des signaux d'entrée et de sortie. Examinons chacun d'eux et définissons leur fonction.

9.4 - Définitions des signaux du CTC

D7-D0 Ce sont les entrées et les sorties de données du composant qui sont reliées au bus de données du Z80. D0 est le bit de poids faible.

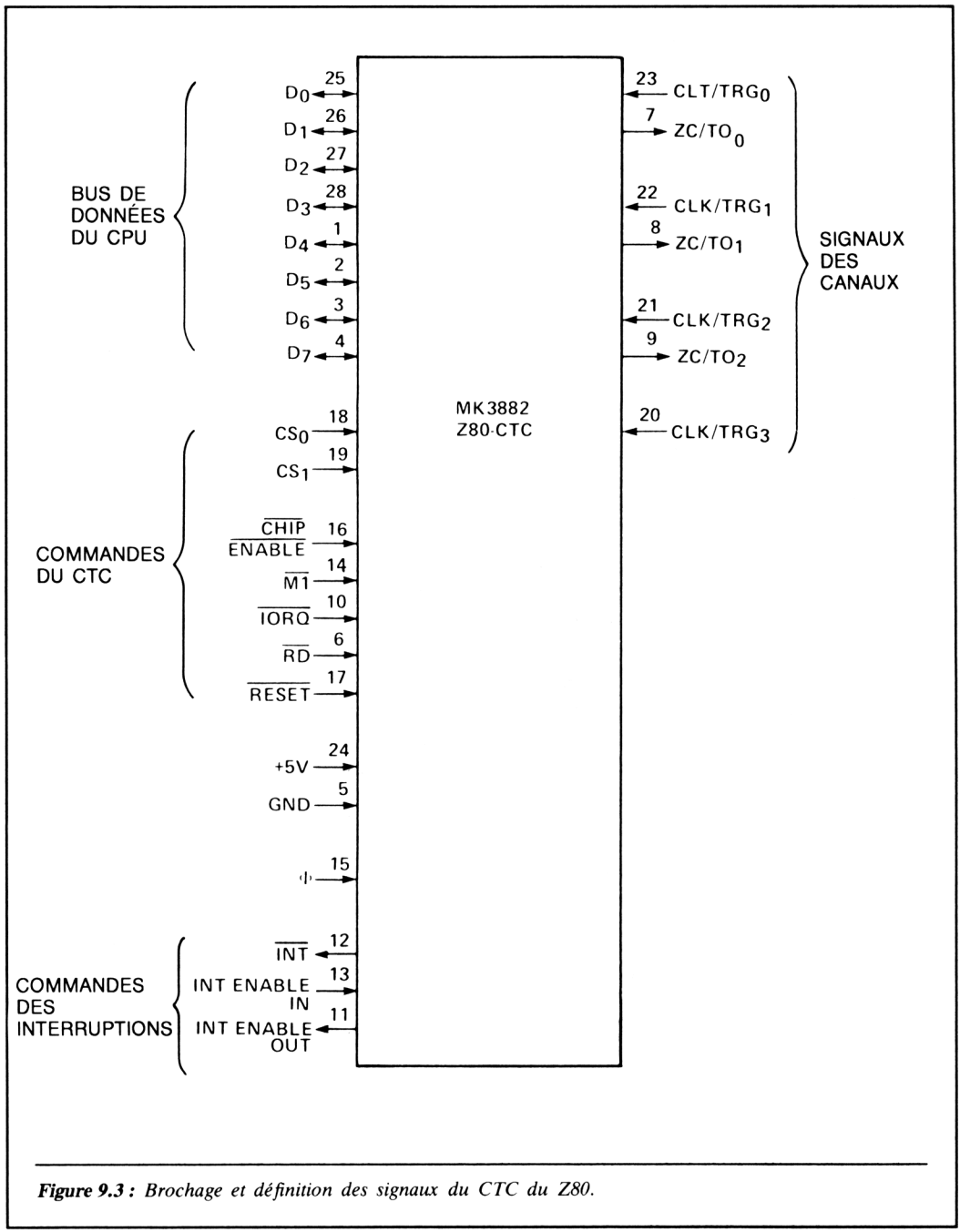


Figure 9.3 : *Brochage et définition des signaux du CTC du Z80.*

CS1, CS0 Ce sont les deux lignes de sélection du canal. Ces deux bits forment l'adresse binaire du canal avec lequel il y a une communication pendant une opération d'entrée-sortie en lecture ou en écriture. Habituellement les lignes A0 et A1 provenant du CPU Z80 sont connectées à ces entrées. La table de vérité pour ces deux entrées est la suivante :

CS1	CS0	CANAL ACTIF
0	0	zéro
0	1	un
1	0	deux
1	1	trois

$\overline{\text{CE}}$ C'est l'entrée de validation du boîtier (*Chip Enable*), active au niveau bas. Quand cette ligne est au 0 logique, le CTC est électriquement capable d'échanger des données avec le Z80. Cette entrée est habituellement une valeur décodée des lignes d'adresses A7 à A2 du bus d'adresses du Z80.

HORLOGE C'est l'entrée d'horloge du système du CPU du Z80. Cette horloge est utilisée par le CTC pour synchroniser certains transferts de données internes.

$\overline{\text{M1}}$ La ligne d'entrée $\overline{\text{M1}}$ est connectée à la sortie $\overline{\text{M1}}$ du microprocesseur Z80. Ce signal sert, avec l'entrée $\overline{\text{RD}}$, à déterminer quand le Z80 cherche un code opération en mémoire. On l'utilise avec l'entrée $\overline{\text{IORQ}}$ pour déterminer quand le Z80 accuse réception d'une demande d'interruption.

$\overline{\text{IORQ}}$ Cette entrée est connectée à la ligne de sortie $\overline{\text{IORQ}}$ du Z80. Elle indique au CTC que le Z80 est en train d'exécuter, en sortie, une opération de lecture ou d'écriture.

$\overline{\text{RD}}$ L'entrée $\overline{\text{RD}}$ est connectée directement à la ligne de sortie $\overline{\text{RD}}$ du Z80. Cette ligne indique au CTC quand le Z80 lit des données en mémoire ou en entrée-sortie. Il faut remarquer que le CTC n'a pas de ligne spécifique d'entrée d'écriture. Pour que le Z80 puisse écrire des données dans le CTC, $\overline{\text{CE}}$ doit être au 0 logique, $\overline{\text{RD}}$ au 1 logique et $\overline{\text{IORQ}}$ au 0 logique.

IEI (*Interrupt Enable Input*, entrée d'autorisation d'interruption - active au 1 logique.) Quand cette entrée est au 1 logique, elle autorise le composant à émettre des demandes d'interruption sur la ligne de sortie $\overline{\text{INT}}$. Quand l'IEI est au 0 logique, la sortie d'interruption est coupée dans le dispositif.

IEO (*Interrupt Enable Output*, sortie d'autorisation d'interruption - active au 1 logique.) Quand cette ligne est au 1 logique, ceci indique que le CTC n'est pas en train de servir une interruption provenant d'un canal interne quelconque. Cette ligne est employée en même temps que IEI pour former un chaînage de priorités d'interruptions, simple et très efficace.

$\overline{\text{INT}}$ C'est la ligne de sortie de demande d'interruption ; elle est à drain ouvert et peut donc être câblée en ET avec d'autres lignes de sortie $\overline{\text{INT}}$ du système.

$\overline{\text{RESET}}$ Un 0 logique sur l'entrée de réinitialisation met le CTC dans un état connu. Dans cet état, tous les canaux arrêtent de compter. Tous les bits d'autorisation d'interruption de tous les registres de commande sont réinitialisés. Ceci empêche le CTC de demander des interruptions au système. Toutes les lignes de sortie ZC/TO et $\overline{\text{INT}}$ sont mises en état de haute impédance. Les *drivers* de sortie des bus de données du CTC sont mis en haute impédance.

CLK/TRG_3-CLK/TRG_0 Ce sont les entrées de déclenchement de temporisateur par horloge externe.

ZC/TO_2-ZC/TO_0 Les sorties compte zéro/fin de temps sont actives à l'état haut.

9.5 - Connexion du CTC au Z80

Maintenant que nous sommes familiarisés avec les différentes entrées et sorties du CTC, connectons-le au microprocesseur Z80. Nous supposons qu'il n'y a pas besoin d'amplificateurs de bus de données. (Si vous ne connaissez pas bien ce point, reportez-vous au Chapitre 2 et à l'analyse des amplifications de bus de données, afin de voir si votre système en nécessite.)

Nous allons commencer l'examen des interfaçages au Z80 en supposant que nous allons utiliser les possibilités d'interruption du CTC, mais il n'est pas nécessaire de les utiliser pour tirer parti de ce composant.

La Figure 9.4 montre le schéma complet des connexions entre Z80 et CTC. Examinons-en plusieurs points importants.

La première liaison à faire entre Z80 et CTC est celle du bus de données. Il faut connecter les lignes D0 à D7 du Z80 aux lignes D0 à D7 du CTC. Ces connexions constituent le moyen physique de passages de données entre Z80 et CTC.

Les lignes de sortie d'adresses A0 et A1 du bus d'adresses Z80 sont reliées aux broches d'entrée CS0 et CS1 du CTC. L'entrée de validation du

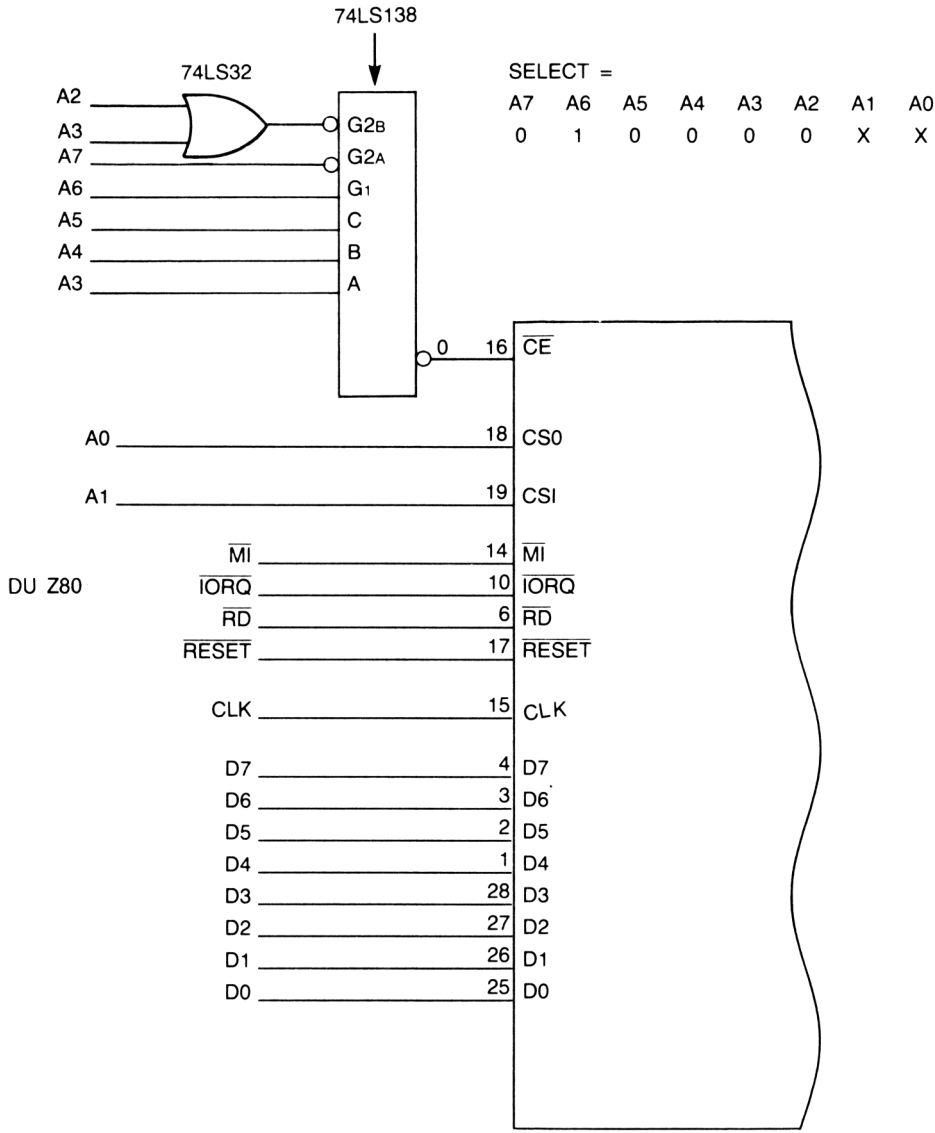


Figure 9.4 : *Schéma montrant les connexions physiques entre CTC et microprocesseur Z80. Le décodage pour l'entrée* $\overline{CS}$ *du CTC est effectué à partir des lignes d'adresses A7-A12.*

boîtier est la valeur décodée obtenue à partir des lignes de sortie d'adresses A2-A7. Dans le schéma de la Figure 9.4, le décodage du CTC est 40H, 41H, 42H et 43H. Remarquez que le décodage complet des quatre ports d'entrée-sortie consiste en une combinaison logique des lignes d'adresses A0-A7.

$\overline{M1}$, $\overline{IORQ}$ et $\overline{RD}$ sont les autres signaux à connecter. Ces trois signaux sont émis vers le CTC. Ils sont reliés directement aux broches du Z80 ayant la même désignation.

L'alimentation et la masse du CTC sont au +5 volts et au 0 volt, l'entrée d'horloge à la broche 15 du CTC est connectée à la broche 6 d'entrée d'horloge du Z80. Il faut remarquer que la fréquence d'horloge maximale applicable est égale à 2,5 mégahertz.

Enfin, la ligne de sortie $\overline{INT}$ du CTC est connectée à la ligne d'entrée $\overline{INT}$ du Z80. Remarquez que la sortie est ramenée au + par l'intermédiaire d'une résistance de 10 kilohms. Il ne faut qu'une seule résistance de rappel pour la ligne d'entrée distincte $\overline{INT}$ du Z80. Ceci est dû au ET des entrées de demandes d'interruption appliquées au microprocesseur.

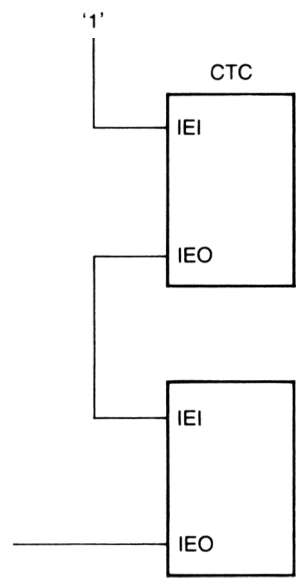


Figure 9.5 : *Schéma montrant comment employer le chaînage de priorités du CTC.*

Si le CTC n'est pas configuré en priorité d'interruption, l'entrée IEI est alors connectée à Vcc. Ceci valide l'interruption au moyen de circuits. La ligne IEO peut être ignorée électriquement (c'est-à-dire laissée non connectée). Si l'on doit connecter le CTC dans la configuration de priorité d'interruption, l'entrée IEI est alors connectée à la sortie IEO de l'autre périphérique. Ceci est montré dans le schéma de la Figure 9.5.

9.6 - Examen du mode compteur du CTC

Maintenant que nous avons connecté le CTC au microprocesseur Z80, nous pouvons fixer notre attention sur l'écriture de logiciel permettant au boîtier de fonctionner de la façon requise. Dans cette section, nous examinerons les principaux points logiciels et matériels permettant d'employer le CTC en mode compteur. La Figure 9.6 montre le schéma du mode compteur du CTC.

On peut voir dans la Figure 9.6 que le CTC est configuré essentiellement pour compter des horloges externes. Celles-ci sont appliquées à la broche

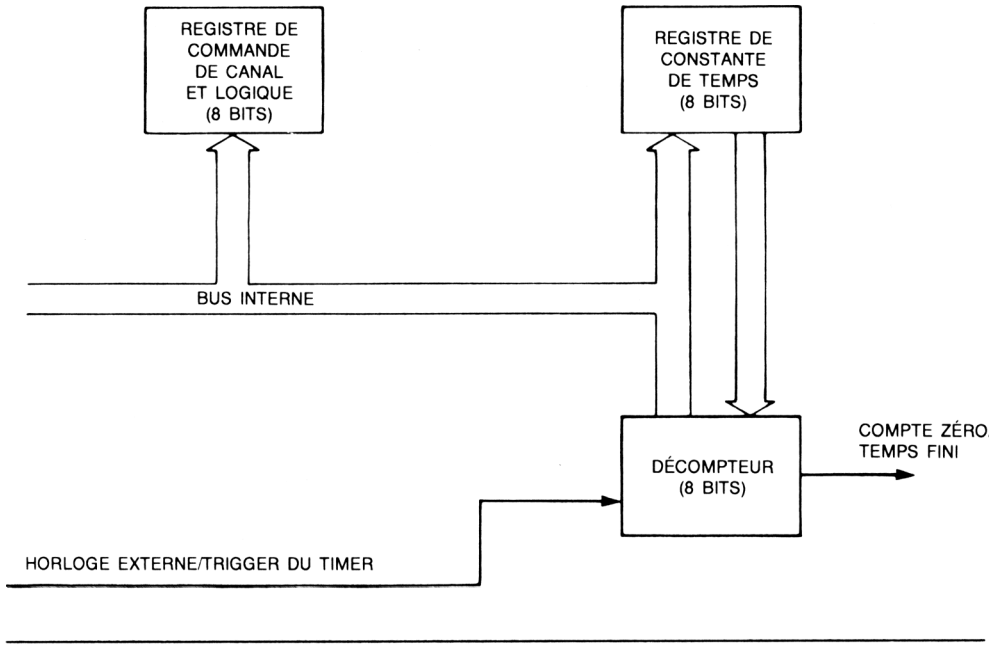


Figure 9.6 : *Schéma de CTC employé en mode compteur.*

d'entrée 5 CLK/TRG du canal sélectionné. On peut, par exemple, employer le CTC pour compter un certain nombre d'événements. Quand on a atteint la valeur pré-établie, le microprocesseur entreprend alors une action donnée.

Pour utiliser le décompteur du CTC, il faut programmer une constante de temps dans le registre de constante de temps. Cette constante doit consister en un nombre binaire d'une valeur maximale de 0FFH. La valeur de la constante de temps sera le nombre initial de départ du décompteur. Dès que celui-ci atteint 00H, la *sortie de compte de zéro* devient active. Nous vous montrerons plus tard la manière exacte de charger le registre de constante de temps. Il suffit, pour l'instant, que vous compreniez ce que l'on peut faire, d'une façon générale, dans le CTC. Vous pourrez plus tard réfléchir à la façon exacte d'y arriver.

Avant que le décompteur ne commence à fonctionner en mode compteur, le registre de commande du canal doit être tenu informé que c'est le mode voulu par le programmeur. Dans ce but, un mot de données doit être écrit dans le registre de commande du canal.

Après l'opération d'écriture dans le registre de commande, le décompteur commence à compter les signaux d'horloge externe. Ils sont échantillonnés sur les fronts montants ou descendants, suivant le mot de programmation envoyé dans le registre de commande.

Un point important concernant l'échantillonnage des signaux d'horloge externe montré dans la Figure 9.5, est que l'horloge est échantillonnée par le décompteur sur le front montant du signal d'horloge du système. Ceci implique qu'on ne peut pas appliquer de signaux d'horloge externe ayant une fréquence supérieure à celle des signaux d'entrée d'horloge du système appliqué à la broche 15 du CTC. En fait, les caractéristiques du CTC limitent la fréquence maximale d'entrée d'horloge externe à la moitié de la fréquence des signaux d'entrée d'horloge du système.

A présent, le compteur est en train de décompter. Dès qu'il atteint 00H, la broche de sortie ZC/TO du canal va au 1 logique pendant environ un cycle de l'horloge du système. Si les interruptions sont validées, le CTC produit une demande d'interruption envoyée au Z80.

Un autre événement qui se produit lorsque le compteur atteint zéro, c'est que le registre de constante de temps est chargé automatiquement à nouveau dans le décompteur. A ce moment-là, la séquence de décomptage redémarre. Il n'y a pas d'arrêt du processus de décomptage. C'est une caractéristique séduisante, car il peut y avoir plusieurs signaux d'horloge externe, avant que l'interruption ne soit servie. Le compteur d'événements va continuer à fonctionner et aucun comptage ne sera perdu.

Si une nouvelle valeur de constante de temps est chargée dans le registre de constante de temps pendant une séquence de décomptage, le comptage en cours continue jusqu'à ce que le zéro soit atteint. A ce moment-là, la

nouvelle valeur de constante de temps est chargée dans le décompteur. Si le programmeur veut terminer le compte et en recommencer un nouveau, il lui suffit d'écrire dans le registre de commande.

On doit écrire dans le registre de constante de temps et dans le registre de commande avant que le CTC ne commence à effectuer la fonction compteur, après mise sous tension ou remise à l'état initial du système.

A tout moment, les données présentes dans le décompteur peuvent être lues par le Z80 pour surveiller les opérations du système. Ceci est effectué par lecture dans le canal sélectionné.

9.7 - Programmation du registre de commande de canal

Examinons maintenant les bits de données de programmation du registre de commande de canal (traité dans la section précédente). Il faut remarquer que certains bits du registre de commande de canal vont être utilisés en mode temporisateur, ce qui n'a pas encore été étudié. Nous verrons ces bits plus en détail, lorsque nous étudierons le mode temporisateur.

La Figure 9.7 montre les bits du registre de commande de canal. Voici l'explication de chaque bit.

D0 Ce bit doit être au 1 logique pour que le CTC interprète les données d'entrée comme mot de registre de commande de canal.

D1 (entrée de remise à l'état initial du canal) Quand ce bit est au 1 logique, le canal cesse de compter ou de mesurer le temps. Aucun des bits du registre de canal ne sera changé. Le canal reprend son fonctionnement normal lorsqu'une constante de temps est à nouveau chargée dans le registre de constante de temps.

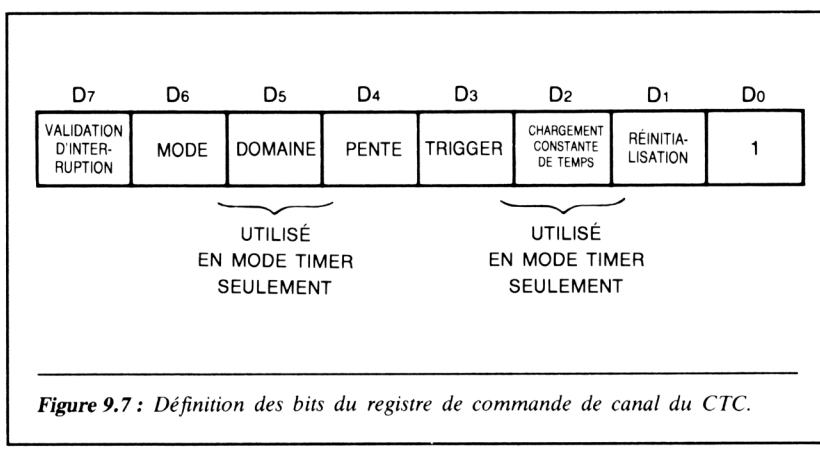


Figure 9.7 : *Définition des bits du registre de commande de canal du CTC.*

D2 Un 1 logique dans ce bit informe le CTC que le mot de canal suivant est une constante de temps. Il faut donc écrire deux mots dans le CTC chaque fois qu'une constante de temps doit être rechargée. Le premier mot est le mot de commande (avec cette configuration de bit) et le second, la valeur de la constante de temps.

D3 Ce bit sert de mode temporisateur seulement. Si c'est un 1 logique, il spécifie que le déclencheur extérieur démarre le temporisateur. Si le bit est un 0 logique, le temporisateur démarre dès que le registre de constante de temps est chargé.

D4 Ce bit détermine le front actif de la broche d'entrée du déclenchement/horloge externe. La définition est :

Mode temporisateur	D4 = 1	Déclenchement par front positif démarre temporisation.
	D4 = 0	Déclenchement par front négatif démarre temporisation.
Mode compteur	D4 = 1	Front positif de décrémentation d'horloge externe.
	D4 = 0	Front négatif de décrémentation d'horloge externe.

D5 Ce bit ne sert qu'en mode temporisateur. Il choisit la valeur de prédivision par 16 ou par 256. Un 1 logique donne une division par 256 ; un 0 logique une division par 16.

D6 Si ce bit est au 1 logique, le canal est sélectionné comme compteur. Le décompteur reçoit comme signaux d'horloge l'entrée CLK/TRG du canal. Si ce bit est au 0 logique, le canal est sélectionné comme temporisateur. Ce décompteur a comme signaux d'horloge l'entrée d'horloge prédivisée. La période de sortie d'impulsions est égale à (période d'horloge du système * facteur de prédivision (16/256) * mot de données de constante de temps).

D7 Quand ce bit est mis au 1 logique, la structure d'interruption interne est validée. Un vecteur d'interruption est écrit dans le Z80 quand la valeur zéro est atteinte par le décompteur. Si, de plus, ce bit est égal à 1, un vecteur d'interruption doit être écrit dans le registre de vecteur d'interruption avant que le canal ne fonctionne correctement. Si ce bit est au 0 logique, les interruptions internes sont inhibées.

9.8 - Programmation du registre de constante de temps

Le canal ne peut pas commencer à fonctionner avant que le registre de constante de temps n'ait été chargé. Ce mot de données sera le prochain mot écrit dans le canal pourvu que le bit D2 du mot de commande ait été mis au 1 logique lors de la précédente opération d'écriture.

Des données 0 à 255 peuvent être écrites dans le registre de constante de temps. Si tous les bits sont au 0 logique, alors la valeur avec laquelle fonctionne le décompteur est 256.

Si un registre de constante de temps est chargé pendant une séquence de comptage en cours, la nouvelle valeur ne sera pas utilisée avant que le compte 0 de la valeur en cours ne soit atteint.

9.9 - Programmation du vecteur d'interruption

La Figure 9.8 montre les bits qui peuvent être programmés pour le vecteur d'interruption. Le CTC est prévu pour emploi avec le Z80, quand il fonctionne en séquence d'interruption de mode 2. On utilise un seul vecteur d'interruption pour tout le boîtier.

Comme le montre la Figure 9.8, le bit D0 est au 0 logique, ce qui informe le CTC que c'est un vecteur d'interruption et non pas un mot de commande.

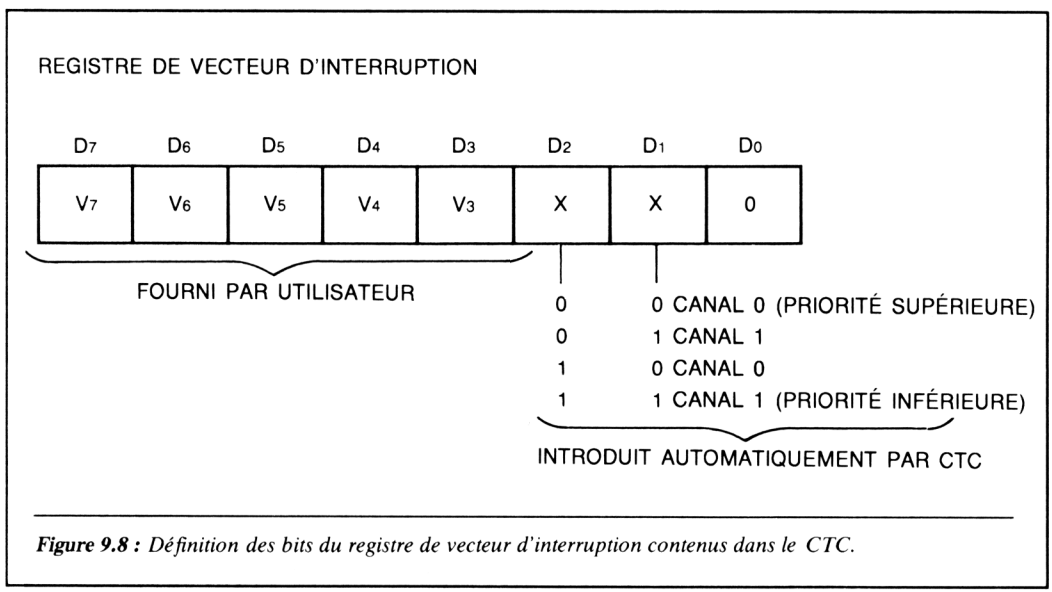


Figure 9.8 : *Définition des bits du registre de vecteur d'interruption contenus dans le CTC.*

Le vecteur est écrit dans le canal 0 seulement. Les bits D1 et D2 sont mis à 1 automatiquement par le CTC quand un accusé de réception d'interruption est produit par le Z80. La combinaison logique de ces deux bits définit distinctement un vecteur pour un canal particulier. De cette façon, l'utilisateur n'a besoin que de charger un seul vecteur d'interruption pour le boîtier entier.

Ce vecteur a les 5 bits de poids fort mis à 1 par l'utilisateur. Ensuite, suivant le canal qui a interrompu le processeur, le CTC place les bits D1 et D2 appropriés dans le vecteur. Le bit D0 est toujours mis au 0 logique. Comme exemple des quatre vecteurs d'interruption, considérons ceci. L'utilisateur a écrit le vecteur d'interruption 58 dans le canal 0.

Ceci configure les bits du vecteur d'interruption ainsi :

```
0 1 0 1 1 0 0 0
```

Les vecteurs d'interruption de canaux correspondants sont alors :

```
canal 0 = 0 1 0 1 1 0 0 0
canal 1 = 0 1 0 1 1 0 1 0
canal 2 = 0 1 0 1 1 1 0 0
canal 3 = 0 1 0 1 1 1 1 0
```

Il est aussi possible que plus d'un canal interrompe le CTC. Il y a une priorité interne, prédéterminée dans le composant. Le canal 0 a la priorité la plus élevée, le canal 3 a la plus basse.

9.10 - Programmation du CTC pour fonctionnement en compteur

Etudions maintenant un exemple d'utilisation du CTC en mode compteur. Nous allons d'abord examiner un système n'utilisant pas d'interruption. Nous examinerons ensuite le même système utilisant des interruptions. En ce qui concerne le premier exemple, reportons-nous au schéma de la Figure 9.9. Nous sommes en train de compter les pannes qui se produisent dans un système en cours d'essai. Chaque fois que le système ne subit pas un essai donné d'une façon satisfaisante, une impulsion est écrite dans le CTC. Voici la définition complète :

1. Le système compte les impulsions. Le front actif est le front montant sur l'entrée d'horloge extérieure du canal.

2. On utilise le canal 1 du CTC.

3. On utilise le CTC en mode de scrutation. Quand le compte atteint 46 en décimal, le microprocesseur effectue une action quelconque.

Pour commencer, définissons les bits du registre de commande. Souvenons-nous d'abord, d'après le schéma de circuits de la Figure 9.4, que les ports d'entrée-sortie du CTC sont égaux à 40H, 41H, 42H et 43H. Comme nous employons le canal 1, nous allons écrire et lire dans le port 41H. Avant d'utiliser le canal 1, nous devons programmer le registre de commande pour définir le fonctionnement du compteur. Voici les bits du registre de commande du canal 1 :

```
bit 7 = 0 Inhibe les interruptions.

bit 6 = 1 Mode compteur sélectionné.

bit 5 = 0 Sans importance car utilisé dans le temporisateur.

bit 4 = 1 Le front montant de l'horloge externe va compter.

bit 3 = 0 Sans importance, car ce bit est utilisé pour le
          temps seulement.

bit 2 = 1 Les données de constante de temps vont suivre.

bit 1 = 1 Réinitialiser le canal, s'il ne fait rien d'autre.
          Souvenez-vous que ce n'est pas une condition
          stockée. Un 1 logique produit une impulsion de
          réinitialisation dans le canal spécifié.

bit 0 = 1 Indication d'un mot de commande et non d'un vecteur
          d'interruption.
```

Le mot est écrit dans le CTC en employant cette séquence d'instructions :

```
LD A,57H
OUT (41H),A     EMETTRE MOT DE COMMANDE VERS CTC
```

Le mot suivant qui doit être écrit dans le port 41H est la constante de temps. On peut écrire d'autres mots dans les ports 40H, 42H et 43H à ce moment. Le CTC s'attend à ce que le prochain mot écrit dans le port 41H soit la constante de temps, car le bit D2 a été mis au 1 logique dans le mot de commande du canal 1.

Dans la définition du problème, nous avons établi que le nombre 46 reçoit la valeur du compte et que cela correspondrait à une valeur hexadécimale de 2EH. Les instructions suivantes vont écrire la constante de temps dans le CTC :

```
LD A,2EH
OUT (41H),A     ECRIRE LA CONSTANTE DE TEMPS DANS LE CTC
```

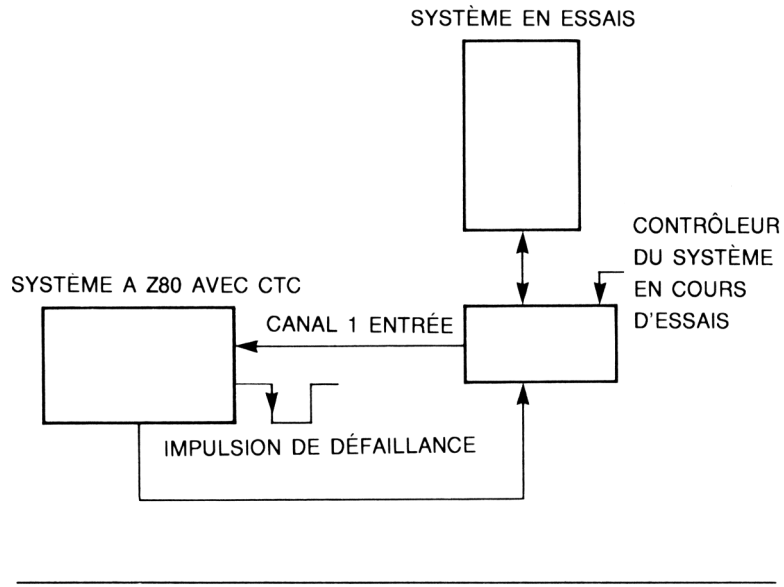


Figure 9.9 : *Schéma montrant comment on peut employer le CTC dans une application type. Ici un système extérieur envoie des impulsions au CTC chaque fois qu'il présente une défaillance au cours d'un essai.*

Aussitôt que le registre de constante de temps est écrit dans le CTC, le canal commence à fonctionner. Voici un programme Z80 complet qui va surveiller le compte en mode de scrutation :

```
        LD A,57H
        OUT (41H),A     ETABLIR MOT DE COMMANDE CH1
        LD A,2EH
        OUT (41H),A     ETABLIR CONSTANTE DE TEMPS = 46 DECIMAL
LOOP    IN A,(41H)      LIRE LE COMPTE
        CP 00H          VERIFIER COMPTE FINAL
        JP NZ,LOOP      SI PAS ZERO CONTINUER A SCRUTER
                        A CE MOMENT-LA, LE Z80 ENTREPREND UNE
                        ACTION, CE QUI INDIQUE QUE LE CTC A
                        REÇU 46 IMPULSIONS CORRESPONDANT A DES
                        DEFAUTS
```

Pendant que le Z80 est en train d'effectuer son action, le CTC recharge automatiquement la constante de temps et recommence à compter.

Présentons maintenant le même programme, mais comportant des interruptions. Après que le Z80 ait configuré le CTC, il entre dans une boucle d'attente d'interruption. Cela, néanmoins, n'a pas besoin d'être le cas dans votre application. On le fait ici à titre d'exemple.

La définition du problème est essentiellement la même. Le mot du registre de commande va changer pour indiquer le fait que l'on utilise des interruptions. Le programme Z80 est le suivant :

```
        DI              INHIBER INTERRUPTIONS INTERNES
;   POUR LE Z80
;   CECI EST EFFECTUE DE TELLE SORTE QUE LE CTC N'EMETTE PAS
;   DE FAUSSES INTERRUPTIONS VERS LE SYSTEME Z80.
;
        IM 2            ETABLIR MODE 2 D'INTERRUPTION
        LD A,80H
        LD I,A          ETABLIR OCTET DE POIDS FORT DU
                        TABLEAU D'INTERRUPTION
;
;
;   DANS CET EXEMPLE, ON SUPPOSE QUE LE TABLEAU D'INTERRUPTION
;   EST SITUE AUX ADRESSES MEMOIRE 8000-80FF. L'ADRESSE DE TA-
;   BLEAU POUR LE CTC EST 8030-8037. 8030, 8031 EST L'ADRESSE DE
;   VECTEUR POUR CH0 (CANAL 0) ; 8032, 8033 EST L'ADRESSE POUR
;   CH1 (CANAL 1) ; 8034, 8035 EST L'ADRESSE POUR CH2 (CANAL 2) ET
;   8036, 8037 EST L'ADRESSE POUR CH3 (CANAL 3).
;
.
        LD A,0D7H       MOT DE COMMANDE POUR CANAL 1
;
;   LE MOT DE COMMANDE PRECEDENT EST LE MEME QUE DANS LE PREMIER
;   EXEMPLE EXCEPTE QUE LE BIT D7 EST MAINTENANT MIS AU 1 LOGIQUE.
;
        OUT (41H),A     ENVOYER LE MOT DE COMMANDE AU CTC
        LD A,2EH        CONSTANTE DE TEMPS = 46 DECIMAL
        OUT (41H),A     ENVOYER CONSTANTE DE TEMPS AU CTC
        LD A,30H        CHARGER VECTEUR D'INTERRUPT. DANS L'ACC.
        OUT (41H),A     CHARGER VECTEUR D'INTERRUPT. DANS LE CTC
        EI              REVALIDER INTERRUPTION
BOUCLE JP BOUCLE        ATTENDRE ICI UNE INTERRUPTION
```

L'adresse de routine d'interruption va être chargée dans les emplacements 8032 et 8033 de la mémoire appropriée. Quand le Z80 revient de la routine de service d'interruption, on doit employer une instruction RETI. Lorsque le CTC détecte cette instruction, la demande d'interruption est ôtée automatiquement de l'entrée $\overline{\text{INT}}$ du Z80.

9.11 - Exemple de fonctionnement du CTC en mode temporisateur

Nous allons montrer dans cette section comment programmer le CTC pour l'utiliser en mode temporisateur type. La Figure 9.10 donne le schéma du CTC en mode temporisateur. Nous allons configurer le CTC en sortie d'horloge donnant des impulsions à la fréquence 2400 Hz. Ce type d'application peut être utile dans un variateur de vitesse de transfert de bits ou dans tout temporisateur du système. On suppose que la fréquence d'entrée de l'horloge du système est égale à un mégahertz.

Cela signifie que nous devons calculer le nombre par lequel il faut diviser la fréquence du signal d'horloge d'entrée pour obtenir une fréquence de 2400 Hz. La Figure 9.11 donne le schéma de ce problème. La période du signal d'horloge à 2400 Hz est égale à 416,7 microsecondes (soit, pour simplifier, 417 microsecondes). La période correspondant à la fréquence du signal d'horloge d'entrée est égale à une microseconde.

Le prédiviseur du CTC doit être mis soit à 16, soit à 256. Si on divise 417 par 16, on a 26,06. Nous l'arrondirons donc à 26. Ce qui signifie que la constante de temps doit être égale à 26. Cela nous donne finalement un diviseur de 16×26, soit 416. Toutes les 416 microsecondes, la sortie

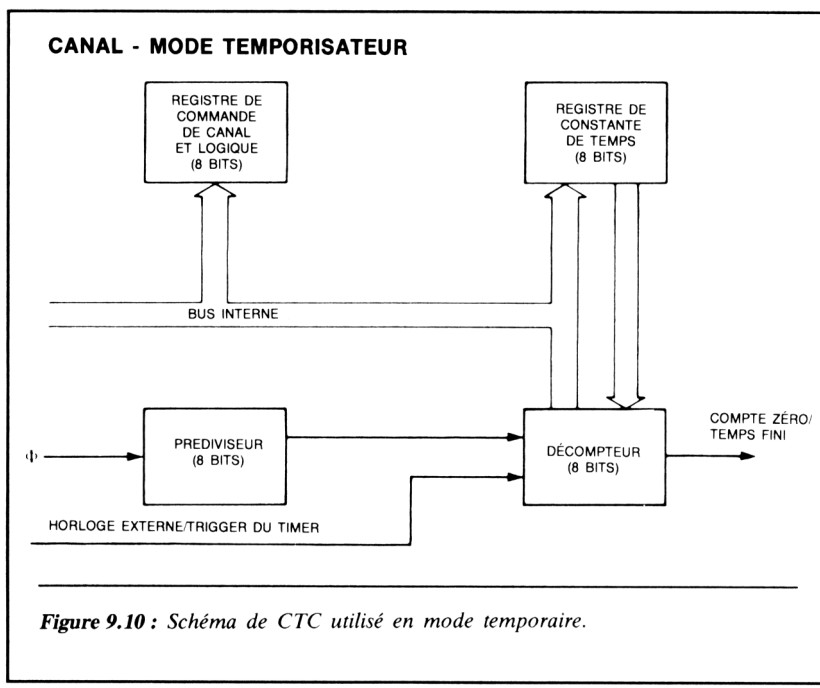


Figure 9.10 : *Schéma de CTC utilisé en mode temporaire.*

ZC/TO va au 1 logique, puis à nouveau au 0 logique. Avec ce chiffre, on peut obtenir une fréquence très proche de 2400 Hz. C'est ce que nous montre la Figure 9.11.

Voyons maintenant la programmation relative à cet exemple. Nous devons d'abord établir le registre du mot de commande. Pour cette application, nous utiliserons le canal 2. Les bits du mot de commande seront comme suit :

D7 = 0 Pas d'interruptions employées dans cette application.

D6 = 0 Mode fonctionnement en temporisateur.

D5 = 0 Facteur de prédivision = 16.

D4 = 0 Sans importance parce que nous n'utilisons pas le *trigger*.

D3 = 0 Le temps démarre quand la constante de temps est chargée.

D2 = 1 Le mot de constante de temps va suivre ce mot de commande.

D1 = 1 Réinitialiser le canal s'il fait quelque chose d'autre.

D0 = 1 Définir cette donnée comme mot de commande.

La donnée de la constante de temps va être égale à 26 (décimal), soit 1AH. On peut programmer le CTC de la façon suivante. On suppose que les circuits sont connectés comme le montre la Figure 9.4.

```
LD A,07H
OUT (042H),A          EMETTRE MOT DE COMMANDE VERS CH2

LD A,1AH
OUT (042H),A          ETABLIR CONSTANTE DE TEMPS,
                      DEMARRER TEMPORISATEUR
```

Le temporisateur fonctionne maintenant sur le canal 2 à une fréquence très proche de 2400 Hz.

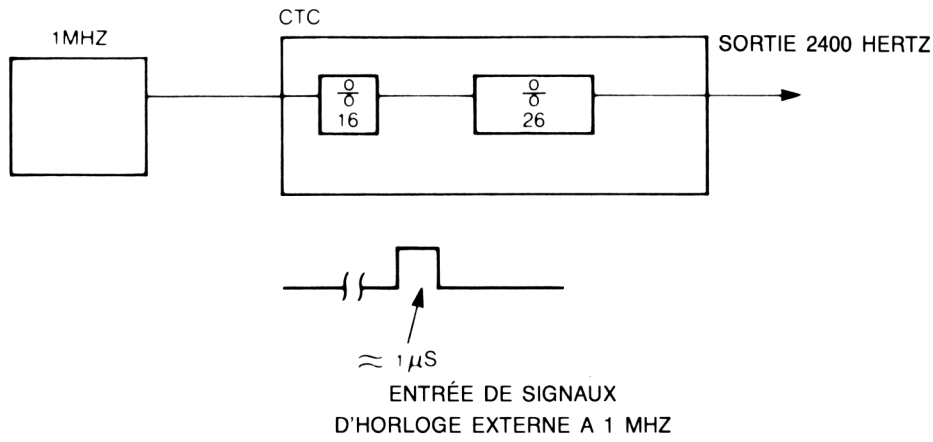


Figure 9.11 : *Schéma du CTC utilisé dans une application de variation de vitesse de transfert de bits. La fréquence de sortie est 2400 Hz avec une fréquence d'entrée de 1 MHz.*

RESUME

Dans ce chapitre, nous avons analysé le fonctionnement et les applications du CTC du Z80. Nous avons d'abord examiné le schéma théorique de ce composant. Nous avons ensuite regardé de plus près les détails nécessaires pour le connecter au microprocesseur Z80, afin d'obtenir une communication fiable. Après avoir connecté les circuits, nous avons porté notre attention sur les points importants de la programmation et de l'utilisation du CTC.

Nous avons exploré, dans ce chapitre, tous les principaux registres internes. Nous avons présenté aussi le logiciel Z80 type permettant de programmer le CTC. Nous avons enfin examiné trois exemples d'utilisation de ce boîtier. Ces exemples ont été choisis de telle sorte que les informations données soient applicables à la plupart des applications incorporant le CTC dans un système à Z80.

Le CTC est un composant offrant beaucoup de souplesse d'emploi ; il peut être précieux en réduisant le nombre de boîtiers de bien des systèmes à Z80. En gagnant de l'expérience dans l'utilisation de ce composant, vous continuerez à découvrir de plus en plus d'applications utiles. Ce chapitre ne constitue qu'un point de départ. Les renseignements que nous venons de vous donner peuvent être employés pour vous aider à connecter et à programmer le CTC d'une façon qui pourra s'appliquer à n'importe quel système.

INTRODUCTION AUX COMMUNICATIONS SERIE

CHAPITRE 10

INTRODUCTION

Dans ce chapitre, nous allons étudier les notions fondamentales des communications série. Nous examinerons un boîtier LSI, le 8251, employé dans les montages destinés aux communications série.

Si vous n'êtes pas très au courant de cette technique, lisez soigneusement ce chapitre. Il est important de bien le comprendre. Après l'avoir étudié, vous devriez avoir acquis une bonne connaissance de base sur la manière de réaliser une communication série dans un système à microprocesseur.

10.1 - Qu'est-ce qu'une communication série ?

Une *communication série* consiste à transmettre des données sous forme d'un flot de bits, un bit à la fois. Cette transmission a lieu sous forme séquentielle dans le temps.

Une *communication parallèle* est l'opposé d'une communication série - tous les bits de données à transférer sont reçus ou transmis en même temps. Un bon exemple de communication parallèle est l'opération de lecture ou d'écriture d'entrée-sortie dans laquelle les huit bits de données sont tous transmis (écrits) ou reçus (lus) en même temps. En fait, dans les communications à microprocesseur que nous avons étudiées jusqu'à présent, toutes les données ont été transmises et reçues en mode parallèle.

Pour mieux représenter ce qu'est une communication série et comment elle diffère d'une communication parallèle, considérons l'exemple suivant. Dans un système à microprocesseur, nous voulons envoyer huit bits de données d'un circuit à un autre. Nous comptons envoyer les données de deux façons : en parallèle et en série.

La Figure 10.1a montre comment apparaissent les données lorsqu'elles sont transmises en mode série ; la Figure 10.1b montre comment elles apparaissent lorsqu'elles sont transmises en mode parallèle. Remarquez que la transmission parallèle nécessite 8 lignes séparées pour la communication, une pour chaque bit à transférer. Dans la transmission série il ne faut qu'une seule ligne physique ; les huit bits de données sont envoyés sur un fil unique, un bit à la fois.

10.2 - Chronologie série

Passons maintenant de l'exemple simple concernant les transmissions série, donné ci-dessus, à quelques notions supplémentaires. Un des points critiques de communication série est la fréquence du flot de bits de données transmis. Cette fréquence s'appelle la *vitesse de transfert*, c'est-à-dire le nombre de bits transmis par seconde en série dans une ligne unique.

Les vitesses de transfert types ont pour valeur 110, 150, 300, 1200, 2400, 4800 et 9600. Supposons que nous voulions transmettre huit bits de données à 2400 bauds. Regardons ce que cela signifie réellement. Les données à

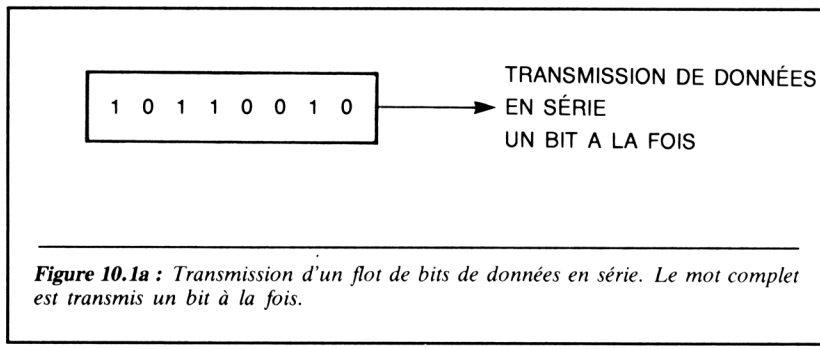


Figure 10.1a : *Transmission d'un flot de bits de données en série. Le mot complet est transmis un bit à la fois.*

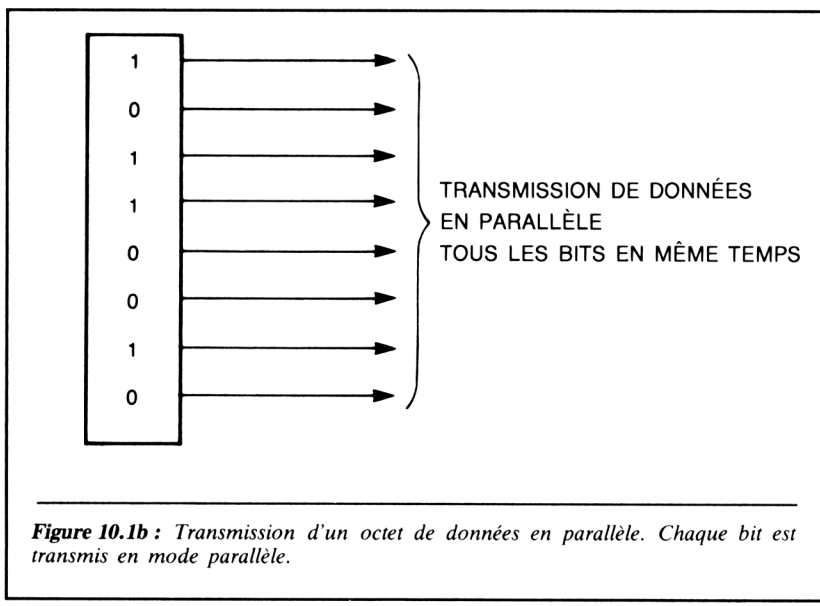


Figure 10.1b : *Transmission d'un octet de données en parallèle. Chaque bit est transmis en mode parallèle.*

transmettre sont montrées Figure 10.2a. La Figure 10.2b montre la forme d'onde que l'on pourrait voir sur un écran d'oscilloscope pendant la transmission des données.

La Figure 10.2b montre aussi que la largeur en secondes d'un des bits transmis est égale à 1/vitesse de transfert. Dans cet exemple, la largeur d'un bit est égale à 1/2400, soit 0,000416 seconde ou 416 microsecondes. Sachant cela, on peut calculer le temps nécessaire pour transmettre la totalité des huit bits. Ce temps est égal à 8 × 416 µs, soit 3.328 µs (ou 3,328 ms). Pour transmettre ces huit mêmes bits en parallèle, il faudrait moins de 1 µs.

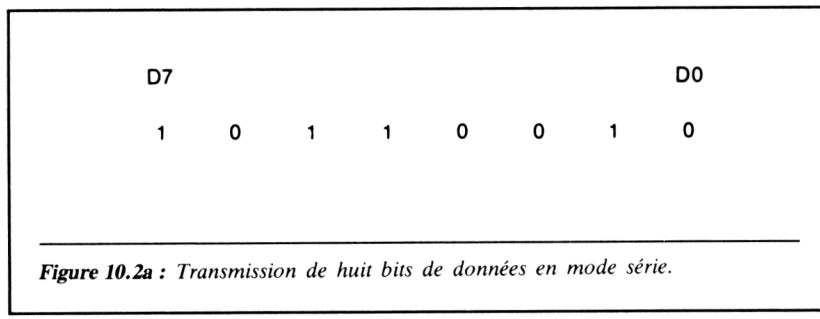


Figure 10.2a : *Transmission de huit bits de données en mode série.*

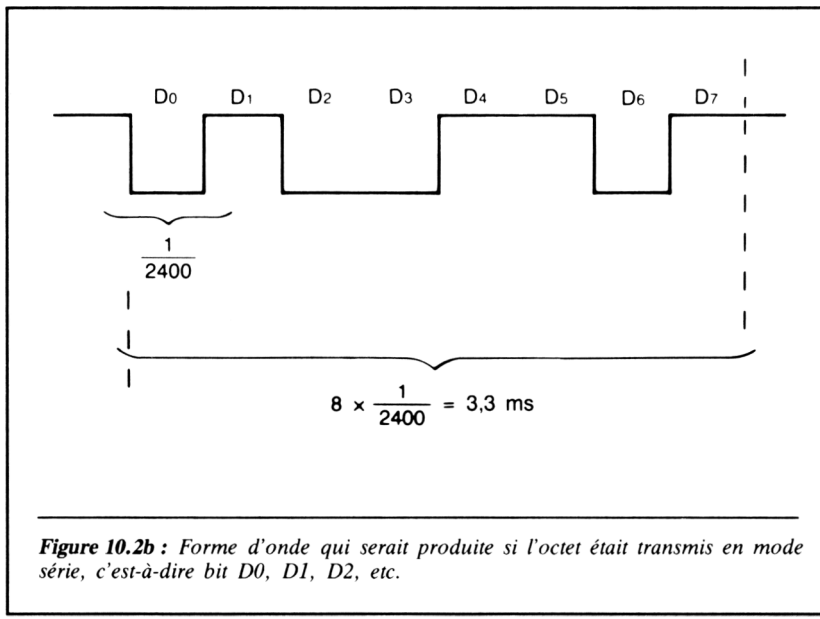


Figure 10.2b : *Forme d'onde qui serait produite si l'octet était transmis en mode série, c'est-à-dire bit D0, D1, D2, etc.*

10.3 - Conversion de données en parallèle en données série

Un point très important concernant la communication série consiste en la conversion de données en parallèle en un flot de bits en série. La communication série doit recevoir les données du microprocesseur qui sont en parallèle et les transformer en un flot de bits. Les différentes étapes de cette conversion sont les suivantes :

1. Stocker le mot de 8 bits en parallèle dans un registre à décalage.

2. Décaler les 8 bits en sortie du registre, un bit à la fois, à la vitesse de transfert appropriée.

Ces deux étapes sont montrées dans le schéma de la Figure 10.3. On peut voir que les données à transmettre sont d'abord produites par le microprocesseur, puis chargées dans le registre à décalage 8 bits à chargement parallèle. Les données sont ensuite décalées en sortie, D0 d'abord, D7 à la fin.

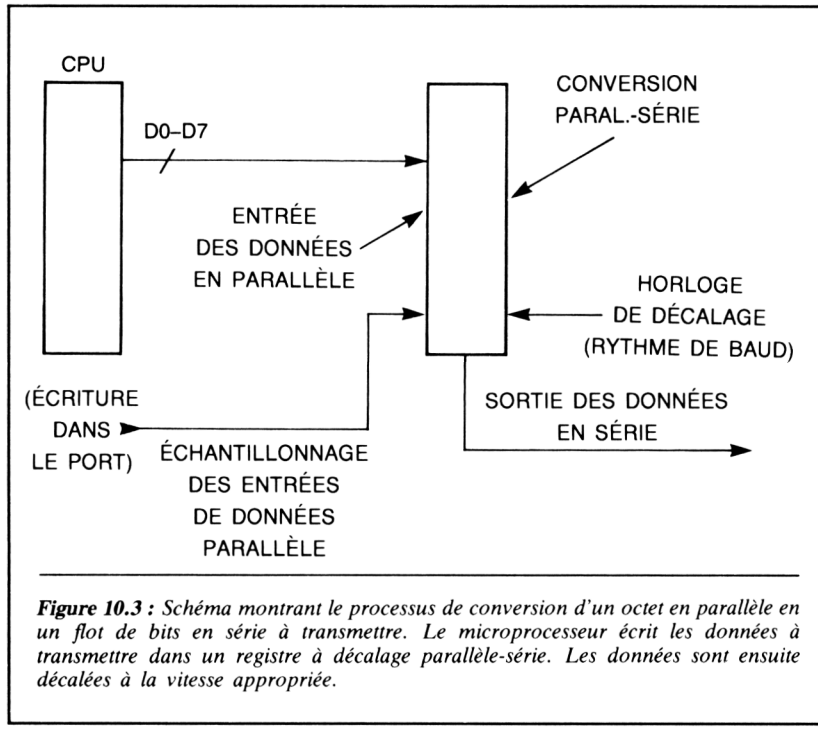


Figure 10.3 : *Schéma montrant le processus de conversion d'un octet en parallèle en un flot de bits en série à transmettre. Le microprocesseur écrit les données à transmettre dans un registre à décalage parallèle-série. Les données sont ensuite décalées à la vitesse appropriée.*

10.4 - Bit de départ

Dans notre exposé sur la transmission série, nous avons expliqué la vitesse de transfert et la conversion parallèle-série. Les données transmises en série doivent pouvoir être reçues et interprétées électriquement. Pour que cela soit possible, un autre bit, appelé *bit de départ (start bit)*, est ajouté automatiquement au flot des bits de données.

La fonction de ce bit est d'informer électriquement le circuit de réception qu'un nouveau flot de données arrive afin de lui permettre de synchroniser son horloge sur le flot de données d'entrée. Chaque flot de données représente un caractère distinct. On peut assimiler le flot de données à un octet des données en parallèle. Bien entendu, les données n'ont pas besoin d'avoir une longueur de huit bits, mais cette manière de penser facilite la compréhension lorsque l'on est novice en matière de transmission série.

Quand la ligne de sortie de transmission de données n'est pas en train d'envoyer des informations, cela correspond à un état appelé *marquage*. C'est l'état d'attente d'une ligne de transmission en série. Supposons que le marquage d'une ligne de transmission soit le 1 logique. Le bit de départ qui est ajouté au flot de bits de données possède l'état logique opposé à celui du marquage. Dans ce cas, le bit de départ sera au 0 logique.

Le bit de départ est en fait un bit ajouté au début du flot de bits de données. Quelle que soit la vitesse de transfert du flot de bits, le bit de départ aura la largeur d'un bit. Ceci est visible en Figure 10.4. Le circuit de réception va détecter ce bit de départ et autoriser l'ensemble récepteur à saisir les nouvelles données.

10.5 - Bit de parité

Un autre bit, qui peut être ajouté à un flot de bits série, est le *bit de parité*. Ce bit est ajouté par l'émetteur et utilisé par le récepteur. Voici ce qu'il fait.

Quand un mot est prêt à être transmis, il contient un certain nombre de 1 logiques. Ce nombre peut être, soit impair, soit pair. Le mot de 8 bits 54H, par exemple, contient trois 1 ; le mot 55H en a quatre. Le circuit de réception est prévu pour recevoir les données et détecter un nombre de 1, soit pair, soit impair.

Supposons que le circuit de réception soit prévu pour recevoir un nombre de 1 qui soit pair à chaque flot de bits série. Le nombre 55H conviendra, mais pas le nombre 54H. Le circuit émetteur va donc ajouter un 1 au flot de bits comportant 54H, avant l'émission de données. Ceci va ajouter un bit supplémentaire au flot des données. Ce sera soit un 1 logique, soit un 0 logique, de façon que le nombre total de 1 soit un nombre pair. La Figure 10.5 montre le bit de parité pour la transmission de 54H et 55H.

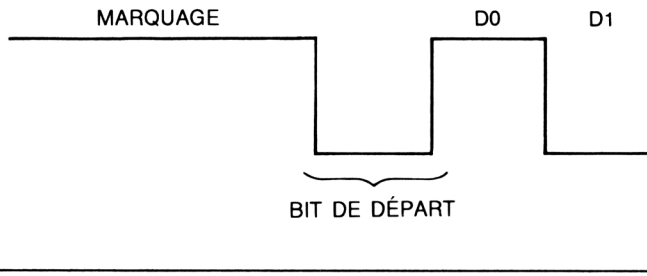


Figure 10.4a : *Diagramme des temps montrant comment apparaîtrait le bit de départ si le premier bit transmis était un 1 logique.*

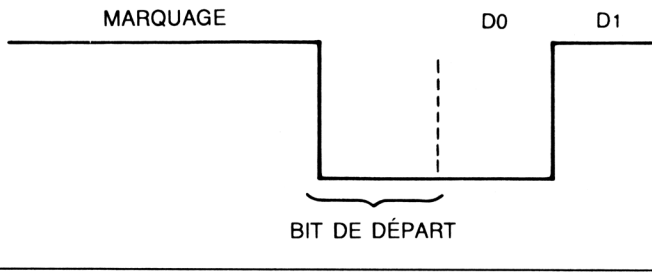


Figure 10.4b : *Diagramme des temps montrant comment le bit de départ apparaîtrait si le premier bit transmis était un 0 logique.*

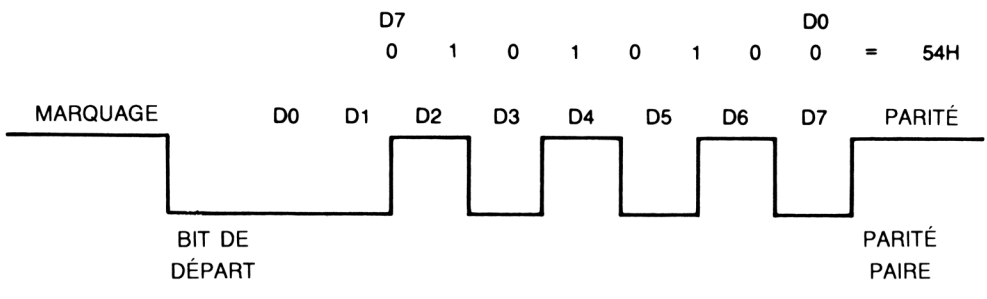


Figure 10.5a : *Diagramme des temps montrant un bit de parité paire pour les données 54H. Le bit de parité est un 1 logique dans ce cas.*

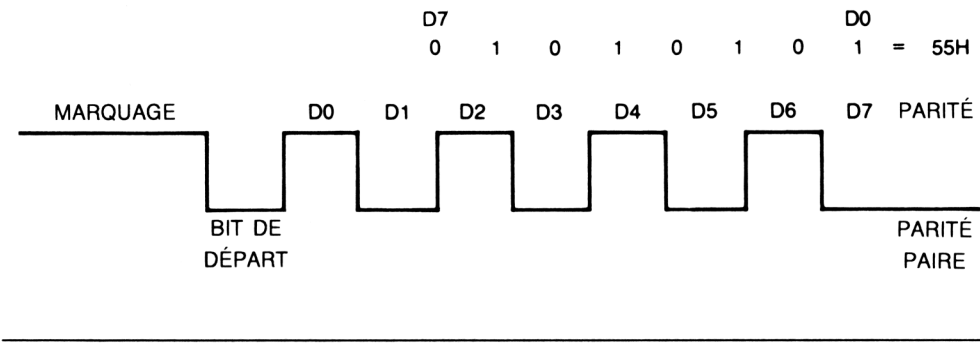


Figure 10.5b : *Diagramme des temps montrant un bit de parité paire pour la donnée 55H. Remarquez que le bit de parité est un 0 logique pour cet octet.*

10.6 - Bit d'arrêt

Le dernier bit ajouté au flot de bits par le circuit de transmission est le *bit d'arrêt (stop bit)*. Le circuit de réception est prévu pour détecter un bit d'arrêt à la fin du flot de données. Il y a plusieurs nombres de bits d'arrêt : 1, 1 1/2 ou 2. La Figure 10.6 montre un mot de données à 8 bits complet avec bits de départ, de données, de parité et d'arrêt. Un flot de données complet consiste ainsi en 12 bits au lieu des 8 bits avec lesquels nous avons commencé. A 2400 bauds, le temps de transmission total des données de la Figure 10.6 est égal à 12×416 µs soit 4,992 ms.

10.7 - Récapitulation des principales notions sur la communication série

Voici, pour récapituler, la liste des principaux points concernant la communication série :

1. Les données série sont transmises un bit à la fois, bit de poids faible d'abord.

2. Les données sont transmises à un rythme fixe appelé vitesse de transfert, c'est-à-dire le nombre de bits transmis par seconde. Par exemple, 1200 bauds correspondent à une transmission de données à une fréquence d'horloge de 1200 hertz.

3. Les données parallèles d'un système à microprocesseur doivent être d'abord converties en données série, pour permettre leur communication en mode série.

4. Le marquage indique l'état logique de la ligne de transmission série, quand elle ne comporte aucune donnée.

5. Le transmetteur de données série ajoute un bit de données, appelé bit de départ, au début des données. Il a l'état logique opposé à celui du marquage de la ligne de transmission série.

6. Le transmetteur peut ajouter un bit distinct, appelé bit de parité, à la fin du flot de données. Ce bit peut produire une parité paire ou impaire. La parité est utilisée par le récepteur comme contrôle d'erreur au premier niveau des données transmises.

7. Le circuit de transmission ajoute des bits d'arrêt au flot de données, après le bit de parité. Le bit d'arrêt a une largeur de 1, 1 1/2 ou 2 bits. Les bits d'arrêt ont le même niveau logique que le marquage de la ligne de transmission série.

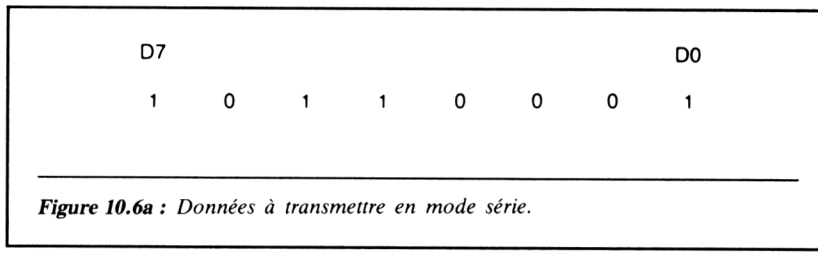


Figure 10.6a : *Données à transmettre en mode série.*

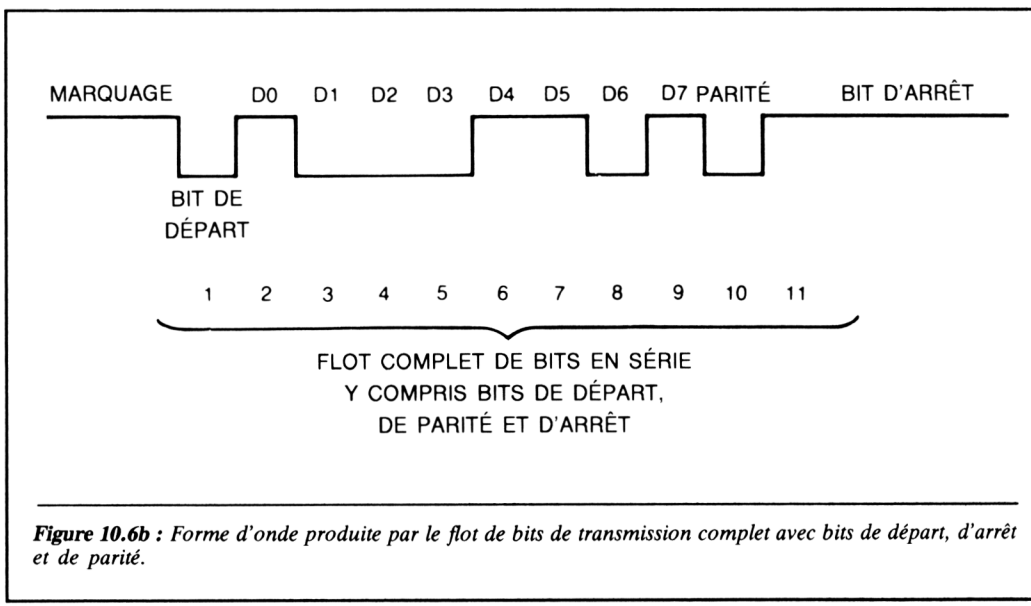


Figure 10.6b : *Forme d'onde produite par le flot de bits de transmission complet avec bits de départ, d'arrêt et de parité.*

10.8 - Examen du 8251

Le principe de transmission de données en série par un système à microprocesseur est souvent employé. Une des utilisations les plus courantes est l'interface ordinateur-terminal. Comme ce type de transmission de données est très répandu, des dispositifs LSI spéciaux ont été créés pour le permettre. Le 8251 en fait partie. La Figure 10.7 montre comment on l'installe dans un système à microprocesseur.

On peut voir dans la Figure 10.7 que le 8251 est traité exactement comme les autres dispositifs d'entrée-sortie spéciaux que nous avons étudiés (comprenant le PIO et le CTC). Le Z80 communique avec le 8251 par le système d'entrée-sortie. Les instructions du Z80 établissent ou programment le 8251 pour qu'il fonctionne d'une certaine façon. Nous allons

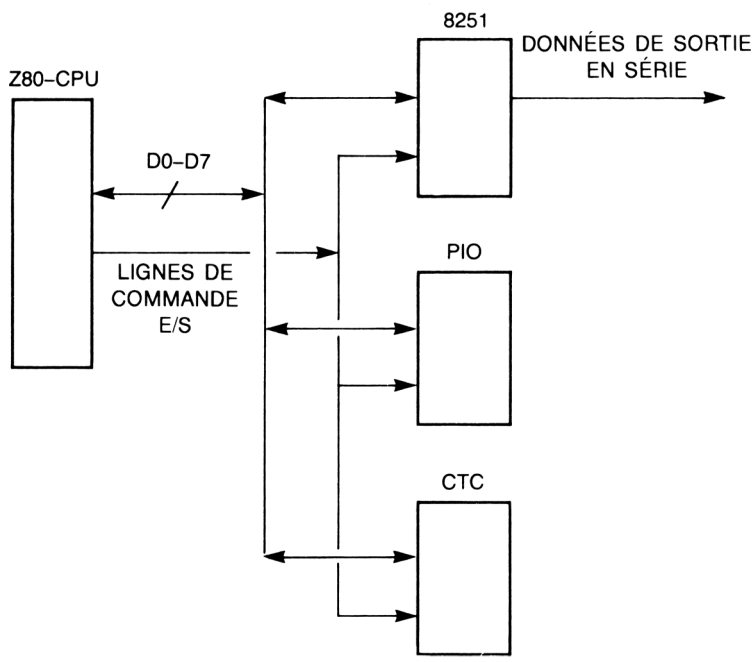


Figure 10.7 : *Schéma montrant comment un 8251 se situe dans un système commandé par Z80.*

montrer dans ce chapitre comment on peut connecter le 8251 à un système à Z80 et obtenir une communication satisfaisante. Analysons, pour commencer, les principales notions de logiciel impliquées dans l'emploi du 8251. Après, vous devriez savoir comment fonctionne un dispositif d'entrée-sortie série type. Ceci vous aidera à comprendre d'autres dispositifs d'entrée-sortie série tels que le SIO du Z80. (Nous étudierons ce composant dans le Chapitre 11.)

La Figure 10.8 montre le schéma du 8251. Analysons la fonction de chaque bloc d'un plan de communication série en commençant par l'*amplification d'adaptation du bus de données*. Ce bloc réalise la liaison physique entre 8251 et bus de données du système à Z80.

Il y a ensuite la *logique de commande lecture/écriture*. Ce bloc assure que la lecture ou l'écriture des données du 8251 est effectuée à l'emplacement interne approprié et avec la chronologie adéquate.

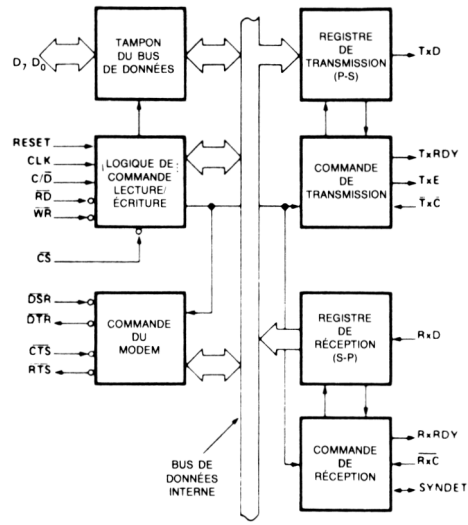


Figure 10.8 : *Schéma de l'architecture interne de l'USART 8251 (Universal Synchronous Asynchronous Receiver Transmitter, récepteur transmetteur synchrone asynchrone universel).*

Le bloc référencé *commande de modem* sert à simplifier l'interface entre 8251 et modem. Les lecteurs qui ne savent pas ce qu'est un modem peuvent noter que c'est un dispositif permettant des transmissions série par ligne téléphonique. La Figure 10.9 montre l'installation d'un modem dans un système de communication série par ligne téléphonique.

En se reportant à la Figure 10.8, on peut voir que les deux blocs référencés *registre de transmission (P-S)* et *commande de transmission* fonctionnent ensemble pour émettre le flot de données en série. (*Note :* P-S signifie conversion parallèle-série.) Les données sont émises par le composant par la ligne référencée TxD. Tx signifie transmission et D données. Le bloc de commande de transmission surveille électriquement l'état du registre de transmission pour déterminer si le registre est vide, c'est-à-dire s'il est prêt à recevoir un autre caractère à transmettre.

Les deux derniers blocs de la Figure 10.8 sont le *registre de réception (S-P)* et la *commande de réception*. Un registre de réception reçoit un flot de données provenant d'un autre dispositif série et le convertit en un mot parallèle qui peut ensuite être lu par le microprocesseur Z80 utilisant une instruction d'entrée. Le bloc de commande de réception surveille l'état du registre de réception pour déterminer électriquement quand il est rempli.

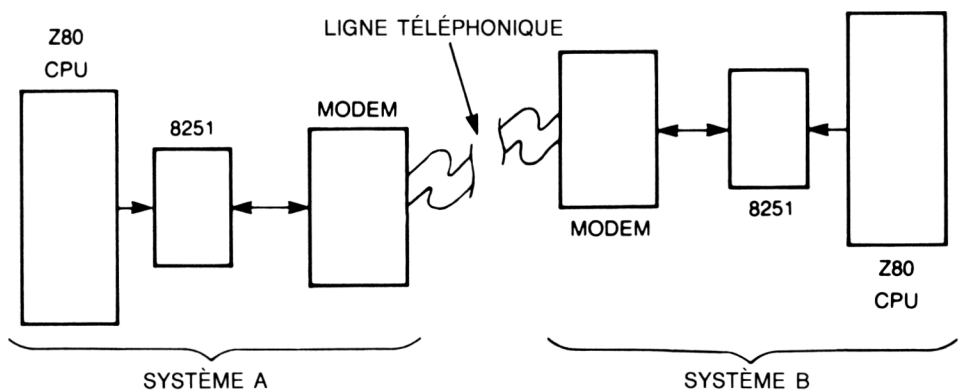


Figure 10.9 : *Ce schéma montre une installation de modem pour transmettre des informations d'un lieu à un autre par ligne téléphonique.*

10.9 - Brochage du 8251

La Figure 10.10 montre le brochage du 8251. Examinons la fonction de chaque broche. Ceci vous aidera à comprendre comment le 8251 peut être connecté physiquement aux bus du système à Z80.

D0-D7 Ce sont les huit lignes de données qui sont reliées au bus de données du système à microprocesseur. Toutes les informations et les mots de programmation sont introduits dans le 8251 ou bien en sortent par ces lignes.

$\overline{\text{RESET}}$ Cette entrée est active au 1 logique. Le 8251 est mis en état d'attente quand $\overline{\text{RESET}}$ est activé. Il reste ensuite dans cet état jusqu'à ce que les mots de programmation appropriés lui parviennent par le bus de données.

CLK (*CLocK*, horloge) Cette entrée du 8251 sert à synchroniser les transferts de données internes. Ce n'est pas l'entrée d'horloge synchronisant la vitesse de transfert. Cette horloge doit être 30 fois plus rapide que la vitesse de transfert du récepteur ou du transmetteur.

$\overline{\text{WR}}$ (*WRite data*, écriture de données - active au 0 logique) Quand cette ligne d'entrée est au 0 logique, les données du bus de données du système sont écrites dans les registres internes du 8251.

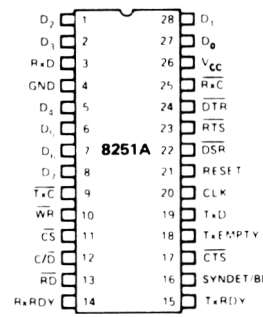


Figure 10.10 : *Brochage de l'USART 8251.*

$\overline{\text{RD}}$ (*ReaD data*, lecture de données - active au 0 logique) Quand cette ligne d'entrée est au 0 logique, les données internes du 8251 sont placées dans le bus de données du système.

C/$\overline{\text{D}}$ (*Control/Data input line*, ligne d'entrée commandes/données) Cette ligne sert à configurer les registres internes du 8251. Quand elle est au 1 logique, les données sont écrites ou lues dans le registre de commandes. Quand elle est au 0 logique, les données sont écrites ou lues dans le registre de données du 8251.

$\overline{\text{CS}}$ (*Chip Select*, sélection de boîtier - active au 0 logique) Un 0 logique dans cette ligne d'entrée autorise l'écriture ou la lecture de données dans le 8251.

COMMANDE DE MODEM Les quatre broches suivantes servent à simplifier l'interface avec un modem :

$\overline{\text{DSR}}$ *Data Set Ready*, jeu de données prêt
$\overline{\text{DTR}}$ *Data Terminal Ready*, terminal de données prêt
$\overline{\text{CTS}}$ *Clear To Send*, mettre à zéro pour émettre
$\overline{\text{RTS}}$ *Ready To Send*, prêt à émettre

Examinons-les en détail.

$\overline{\text{DSR}}$ Cette entrée peut être testée par le CPU. Elle sert habituellement à tester les conditions du modem, telles que *Data Set Ready*.

$\overline{\text{DTR}}$ Cette sortie peut être mise à l'état bas en envoyant une certaine configuration de bits au 8251. Elle sert habituellement à Data Terminal Ready.

$\overline{\text{CTS}}$ Un 0 logique à cette entrée autorise le 8251 à transmettre des données en série, si les conditions internes du composant le permettent. La ligne CTS peut servir aussi de ligne de *handshaking* hardware.

$\overline{\text{RTS}}$ Ce bit de sortie peut être mis à l'état bas en écrivant un mot approprié dans le 8251.

TxD (*Transmit Data output*, sortie de transmission de données) C'est la ligne de sortie par laquelle sont transmises les données en série.

RxD (*Receive Data input*, entrée de réception de données) C'est la ligne d'entrée par laquelle sont reçues les données lorsqu'elles vont au 8251 en flot de bits série.

TxC (*Transmit baud rate Clock input*, entrée d'horloge régulant la vitesse de transmission en sortie).

RxC (*Receive baud rate Clock input*, entrée d'horloge régulant la vitesse de transmission en réception.)

TxRDY Cette sortie signale au CPU que le 8251 est prêt à accepter un autre caractère à transmettre. Cette ligne peut servir de ligne d'interruption au CPU.

TxEMPTY Quand le 8251 n'a pas de caractère à transmettre, cette ligne va au 1 logique. Elle est mise automatiquement au 0 logique quand le CPU écrit un caractère dans le registre du transmetteur.

10.10 - Connexion du 8251 aux bus du Z80

Nous allons maintenant apprendre à connecter le 8251 au microprocesseur Z80. Nous n'allons pas entreprendre de le programmer pour le moment. Si le Z80 ne peut pas échanger correctement des données avec le 8251, il n'est pas intéressant de savoir le programmer correctement, car les informations de programmation ne l'atteindront pas électriquement.

La Figure 10.11 donne un schéma complet montrant comment le 8251 se relie aux bus du système à Z80. La connexion de sortie série du composant n'est pas montrée cette fois-ci. La Figure 10.11 montre un tampon de données entre bus de données du 8251 et bus de données du système à Z80. Ce tampon peut, ou non, être nécessaire dans votre application. S'il n'est pas nécessaire, les lignes de données D0-D7 du 8251 peuvent alors être connectées directement aux lignes de données D0-D7 du bus de données du Z80.

L'entrée $\overline{CS}$ du 8251 résulte du décodage des lignes du bus d'adresses A1-A7. Dans cet exemple, le code de port est 7CH et 7DH. Ces deux codes indiquent que la ligne d'adresse A0 du Z80 est connectée à la ligne d'entrée C/$\overline{D}$ du 8251. $\overline{CS}$ est actif lorsque A0 est soit au 0 logique, soit au 1 logique.

Les entrées $\overline{RD}$ et $\overline{WR}$ sont connectées aux lignes de commandes $\overline{IOR}$ et $\overline{IOW}$ produites par le Z80 comme partie du bus de commandes du système. L'entrée $\overline{RESET}$ du 8251 est active au 1 logique. C'est le niveau logique opposé à celui ayant l'état actif de l'entrée $\overline{RESET}$ du Z80. La Figure 10.11 montre que les liaisons entre 8251 et bus du système à Z80 sont semblables aux connexions entre bus et autres périphériques (ceci a été étudié dans les chapitres précédents).

Les signaux d'horloge pour la vitesse de transfert sont appliqués aux broches TxC et RxC du 8251. Ces horloges n'ont pas besoin d'avoir la même fréquence. Le 8251 est électriquement capable de recevoir et de

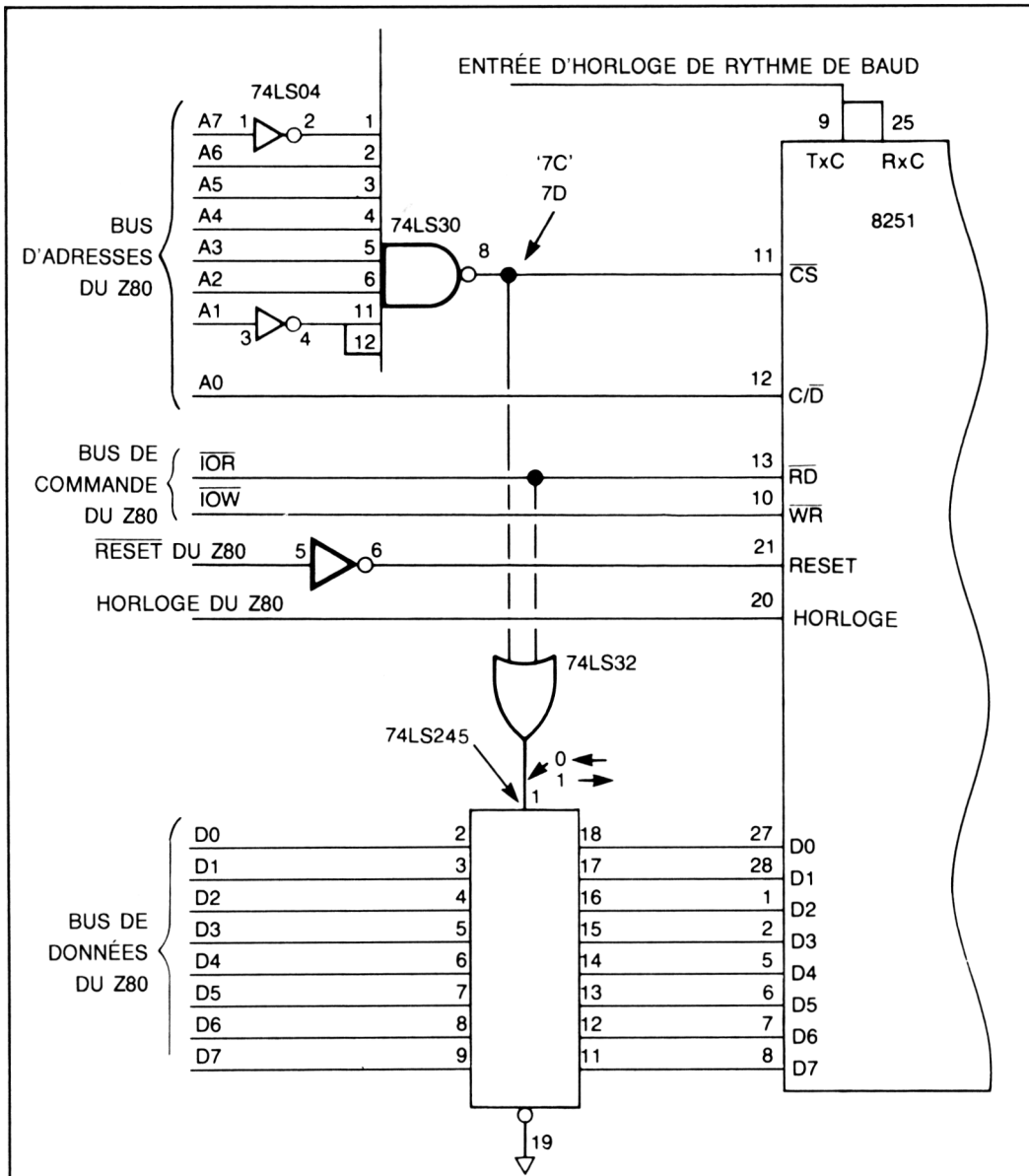


Figure 10.11 : *Schéma complet montrant une façon de connecter le 8251 aux bus du système à Z80. Remarquez l'utilisation d'amplificateurs de données entre le bus de données du Z80 et les broches d'entrées de données du 8251.*

transmettre des données à différentes vitesses. Néanmoins, l'horloge du 8251 et celle du dispositif série associé, qui fait partie de la communication, doivent être synchronisées.

Une autre entrée d'horloge du 8251, qui doit être fournie extérieurement, est appliquée à la broche 20. Cette horloge sert à la synchronisation de transferts de données internes dans le composant. Les spécifications correspondant à cette horloge indiquent qu'elle doit être au moins 30 fois plus rapide que les entrées d'horloge de transmission de récepteur, TxC ou RxC. Si par exemple, on transmet ou on reçoit à 2400 bauds, l'horloge d'entrée RxC et TxC est à 2400 Hz et la fréquence d'entrée d'horloge à la broche 20 devra être égale à 30 × 2400 Hz, soit 72 kilohertz.

10.11 - Connexions série

Connectons maintenant le 8251 aux lignes de transmission série et de réception série. Il existe plusieurs standard de transmission série. Pour cet exemple, nous allons en choisir un qui est très répandu actuellement : le standard électrique RS-232 pour transmission série.

Le RS-232 emploie des impulsions à tension élevée au lieu des niveaux TTL normaux. Les niveaux de tension de la ligne de transmission RS-232 ont approximativement ± 12 volts. La Figure 10.12 donne le schéma d'une interface type avec le bus RS-232. Cette figure montre que le niveau TTL du 8251 doit être amené au niveau du RS-232 avant transmission par la ligne.

A l'extrémité côté réception des lignes RS-232, les niveaux logiques doivent être reconvertis en niveaux TTL avant d'être appliqués au récepteur série. La conversion des niveaux de tension TTL en niveaux de tension RS-232 est si usuelle que des circuits intégrés spéciaux ont été créés pour effectuer cette fonction. On peut citer par exemple, le MC1488 et le MC1489. Les Figures 10.13 et 10.14 montrent une partie des feuilles de spécifications de ces composants.

La Figure 10.15 montre comment on emploie le MC1488 et le MC1489 dans un système type effectuant une communication série RS-232. Remarquez que pour une communication complète entre dispositifs série, il ne faut que trois fils : Tx (transmission), Rx (réception) et masse.

Cette analyse de circuits a présenté les principaux points de la transmission en série que l'on doit avoir présents à l'esprit lorsque l'on envisage d'utiliser la communication série. Dans un système employant la communication série, les blocs hardware que nous avons montrés doivent exister d'une façon ou d'une autre. Du moment que vous savez ce qu'il y a dans ces blocs, il devient beaucoup plus facile d'explorer et d'analyser les circuits.

10.12 - Programmation du 8251

Maintenant que nous avons connecté le 8251 au Z80, concentrons-nous sur le logiciel nécessaire au fonctionnement de ce composant dans un ensemble communication série asynchrone. D'une façon générale, le logiciel indique les paramètres de transmission au 8251. Ils comportent la vitesse de transfert, le nombre de bits de données, le nombre de bits d'arrêt et l'information de parité.

Une fois la configuration établie dans le 8251, la transmission effective a lieu. Le microprocesseur détermine d'abord si le registre de transmission est

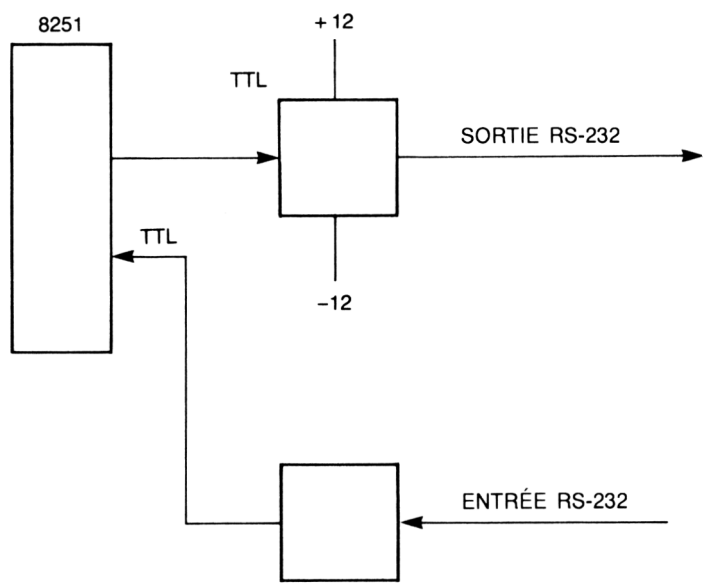


Figure 10.12 : *Schéma montrant comment les niveaux de tension TTL de l'USART 8251 sont convertis en niveaux de tension RS-232. On voit aussi le convertisseur des niveaux de tension RS-232 en niveaux TTL pour l'USART.*

MOTOROLA

MC1488

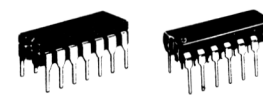
QUAD LINE DRIVER

The MC1488 is a monolithic quad line driver designed to interface data terminal equipment with data communications equipment in conformance with the specifications of EIA Standard No. RS-232C.

Features:

- Current Limited Output
  ±10 mA typ
- Power-Off Source Impedance
  300 Ohms min
- Simple Slew Rate Control with External Capacitor
- Flexible Operating Supply Range
- Compatible with All Motorola MDTL and MTTL Logic Families

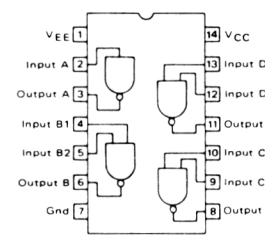
QUAD MDTL LINE DRIVER
RS-232C
SILICON MONOLITHIC
INTEGRATED CIRCUIT



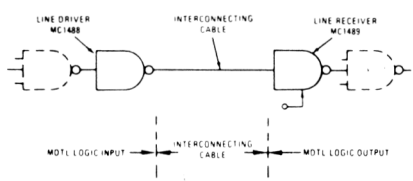
L SUFFIX
CERAMIC PACKAGE
CASE 632
TO-116

P SUFFIX
PLASTIC PACKAGE
CASE 646

PIN CONNECTIONS



TYPICAL APPLICATION



CIRCUIT SCHEMATIC
(1/4 OF CIRCUIT SHOWN)

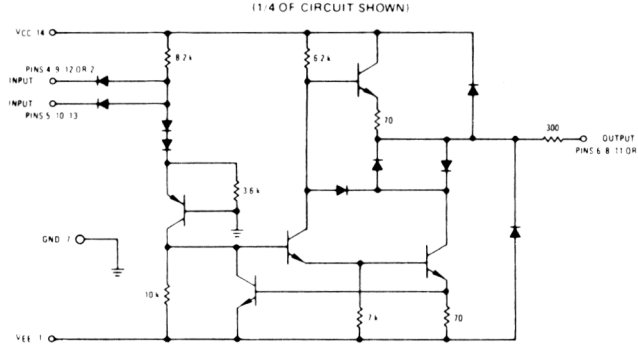


Figure 10.13 : Feuille de données partielle de l'adaptateur de tension Motorola MC1488 TTL à RS-232.

MOTOROLA

MC1489L
MC1489AL

QUAD LINE RECEIVERS

The MC1489 monolithic quad line receivers are designed to interface data terminal equipment with data communications equipment in conformance with the specifications of EIA Standard No. RS-232C.

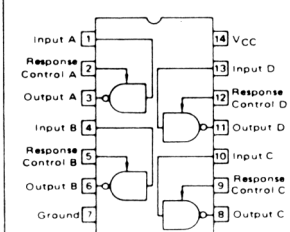
- Input Resistance – 3.0 k to 7.0 kilohms
- Input Signal Range – ± 30 Volts
- Input Threshold Hysteresis Built In
- Response Control
  a) Logic Threshold Shifting
  b) Input Noise Filtering

QUAD MDTL LINE RECEIVERS RS-232C

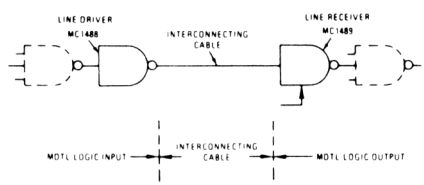
SILICON MONOLITHIC INTEGRATED CIRCUIT

L SUFFIX
CERAMIC PACKAGE
CASE 632
TO-116

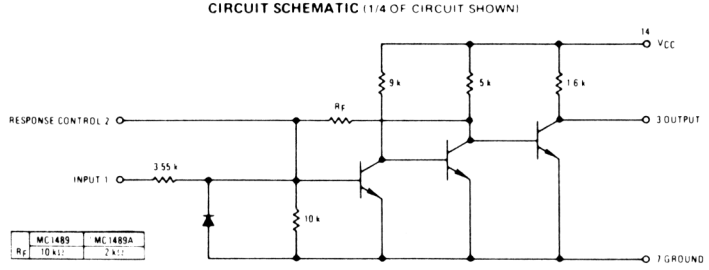
P SUFFIX
PLASTIC PACKAGE
CASE 646



TYPICAL APPLICATION



CIRCUIT SCHEMATIC (1/4 OF CIRCUIT SHOWN)



	MC1489	MC1489A
R_F	10 kΩ	2 kΩ

Figure 10.14 : *Feuille de données partielle de l'adaptateur de tension Motorola MC1489 RS-232 à TTL.*

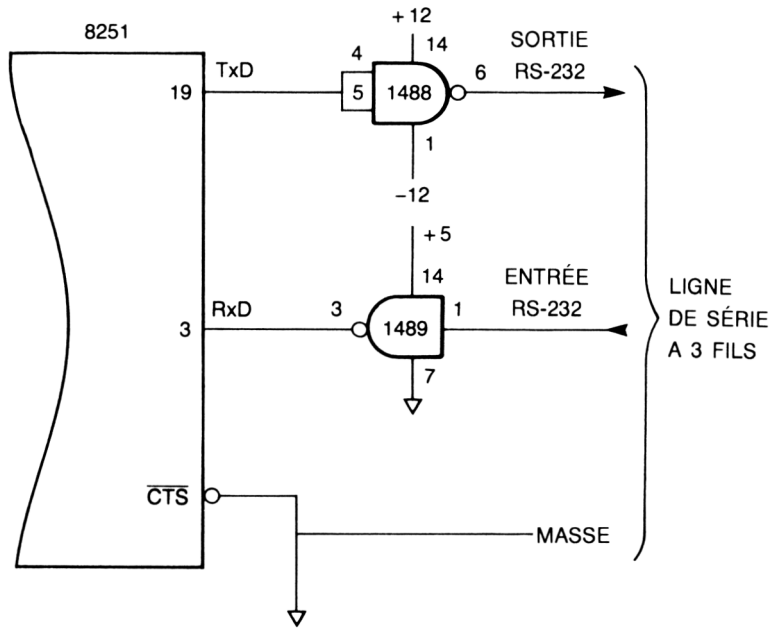


Figure 10.15 : *Schéma montrant comment on peut utiliser les boîtiers MC1488 et MC1489 pour une communication série.*

vide. S'il l'est, un caractère est alors chargé dans le registre et émis dans la ligne de transmission série. Ceci a lieu pour tous les caractères transmis par le CPU.

Pour que le CPU reçoive un caractère, il doit examiner d'abord un registre d'état pour déterminer si le récepteur va lire le caractère. Il doit ensuite attendre que le registre d'état indique quand un autre caractère a été reçu.

La description ci-dessus des réceptions et transmissions série de caractères ne donne qu'une version simplifiée de ce qui se passe réellement. Nous allons maintenant expliquer comment on peut effectuer ces opérations au moyen du 8251.

Avant la transmission, un jeu de mots de commande doit être chargé dans le 8251. Dans ce but, le CPU doit écrire dans le registre de commande immédiatement après activation de la mise à l'état initial du système. Le 8251 sait, physiquement, que la première écriture d'entrée-sortie dans le

registre de commande est l'instruction de mode et qu'un mot de commande va suivre immédiatement. Le mot de commande comporte un ou plusieurs octets. Le nombre d'octets dépend de l'octet envoyé pour l'instruction de mode.

Le schéma de la Figure 10.16 montre les attributions des bits du mot de mode. Utilisons-le pour établir un mot de mode pour une transmission type. Pour cette transmission, nous allons définir notre ligne série ainsi :

1. La vitesse de transfert est égale à 2400 bauds. Notre générateur de vitesse doit avoir la même fréquence que les données transmises. Les bits D1 et D0 du mot de mode vont donc être mis à 0 et 1, respectivement. On peut appliquer des signaux d'horloge 16 ou 64 fois plus rapides que la vitesse de transfert spécifiée. Dans ce cas, le facteur de la vitesse de transfert 16X ou 64X est programmé dans le 8251. Les signaux d'horloge d'entrée Tx ou Rx vont, en effet, être divisés par ce facteur de vitesse de transfert avant d'aller dans le récepteur ou le transmetteur interne du 8251.

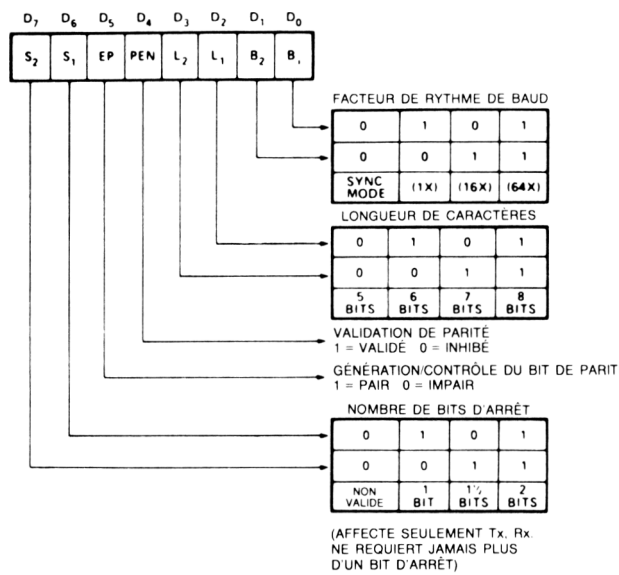


Figure 10.16 : *Schéma de définition du mot de mode du 8251.*

2. Il doit y avoir 8 bits de données transmis pour chaque caractère.

3. Le bit de parité doit être établi pour la parité paire.

4. Le transmetteur doit introduire 2 bits d'arrêt par caractère.

Avec ces spécifications, le mot de mode écrit dans le 8251 va être :

```
1 1 0 1 1 1 0 1
```

Décomposons ce mot et examinons-le de plus près :

```
Bits D7, D6 = 1 1.      Ceci établit 2 bits d'arrêt.

Bits D5, D4 = 0 1.      Ceci établit une parité paire et
                        valide la parité.

Bits D3, D2 = 1 1.      Ces bits mettent la longueur du
                        caractère à 8 bits.

Bits D1, D0 = 0 1.      Le facteur de vitesse de transfert
                        est X1.
```

Les instructions Z80 permettant d'écrire ce mot sont :

```
LD A,0DDH

OUT (7DH),A             SORTIE VERS LE REGISTRE DE COMMANDE
```

Le mot suivant écrit dans le 8251 est l'instruction de commande. Les définitions des bits de cet octet sont montrées en Figure 10.17. En se basant sur cette figure, l'octet écrit est :

```
0 0 0 1 0 1 0 1
```

Cet octet valide le récepteur et le transmetteur et remet à zéro les registres d'erreur. A ce moment-là, le 8251 est prêt pour la réception ou la transmission de caractères.

Le dernier registre dont nous allons parler maintenant, est le *registre d'état.* Ce registre est à lecture seulement. On l'utilise afin que le microprocesseur puisse surveiller l'état de la transmission série et vérifier

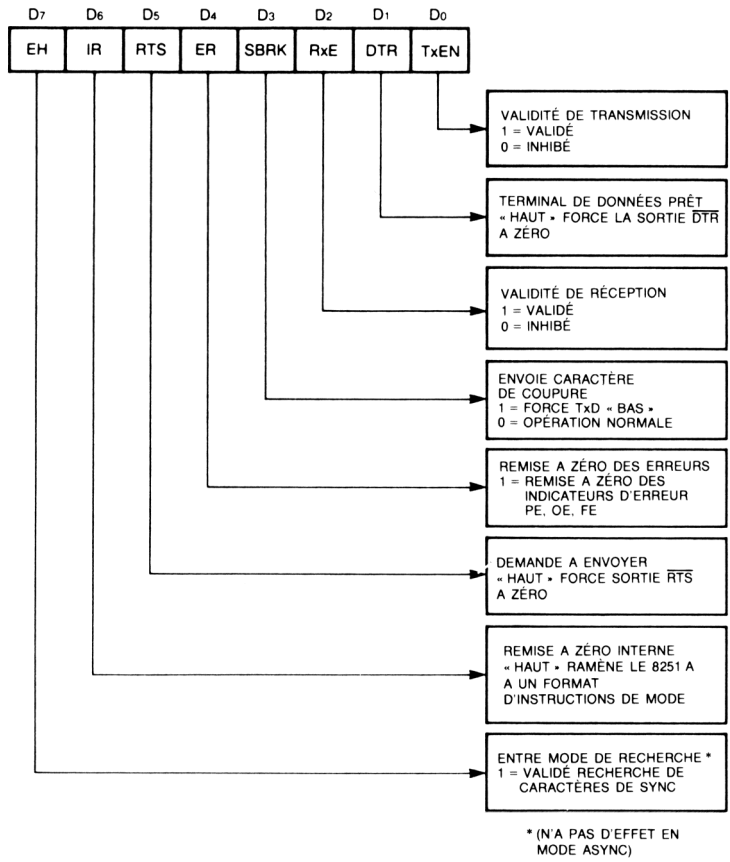


Figure 10.17 : *Définition des bits du format de mot d'instructions de commande du 8251.*

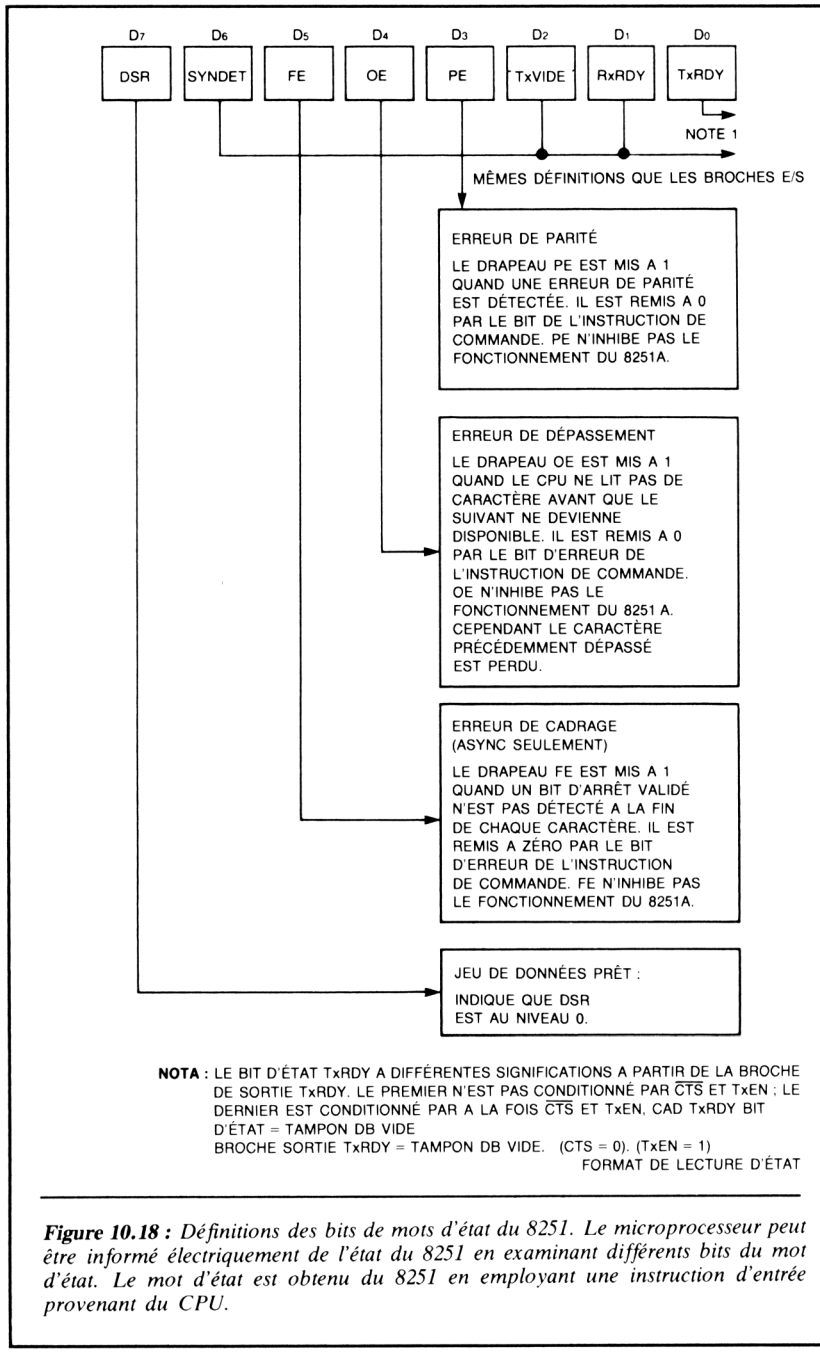


Figure 10.18 : *Définitions des bits de mots d'état du 8251. Le microprocesseur peut être informé électriquement de l'état du 8251 en examinant différents bits du mot d'état. Le mot d'état est obtenu du 8251 en employant une instruction d'entrée provenant du CPU.*

les erreurs de transmission détectables par le 8251. Les définitions de bits du registre d'état sont montrées par la Figure 10.18. On peut lire le registre d'état par une simple opération d'entrée de lecture, assurant ainsi que l'entrée C/$\overline{D}$ du 8251 est au 1 logique.

10.13 - Erreur de cadrage

Si vous ne connaissez pas très bien la transmission série, une explication de certaines erreurs associées à ce type de communication peut être utile. Le premier type d'erreur à examiner est l'*erreur de cadrage*. Elle a lieu quand un caractère est reçu dans le registre tampon d'entrée série et qu'aucun bit d'arrêt n'est détecté dans le flot de données. Ce type d'erreur provoque habituellement une lecture de fausses données.

10.14 - Erreur de dépassement

Une *erreur de dépassement* a lieu lorsqu'un caractère est reçu dans le registre tampon d'entrée série et que le caractère précédent n'a pas encore été lu par le CPU. Ceci provoque la perte du caractère précédent, le nouveau caractère étant écrit sur lui.

10.15 - Programme d'application simple pour 8251

Ce qui suit est un programme Z80 simple pour utilisation d'un 8251. Ce programme envoie d'une façon répétitive le même caractère au terminal. Il n'est pas très utile, excepté qu'il montre certains points importants concernant la transmission avec scrutation. La Figure 10.19 donne l'ordinogramme de ce programme. Les mnémoniques du Z80 qui permettent d'exécuter cet ordinogramme sont montrés dans le programme de la Figure 10.20.

10.16 - Développement d'une application pour 8251

Examinons maintenant un programme Z80 qui permet au 8251 de renvoyer en écho un caractère qui lui est envoyé par un terminal. Ce programme montre comment fonctionne le 8251 pour la transmission et la réception des caractères. La Figure 10.21 montre le schéma des intercon-

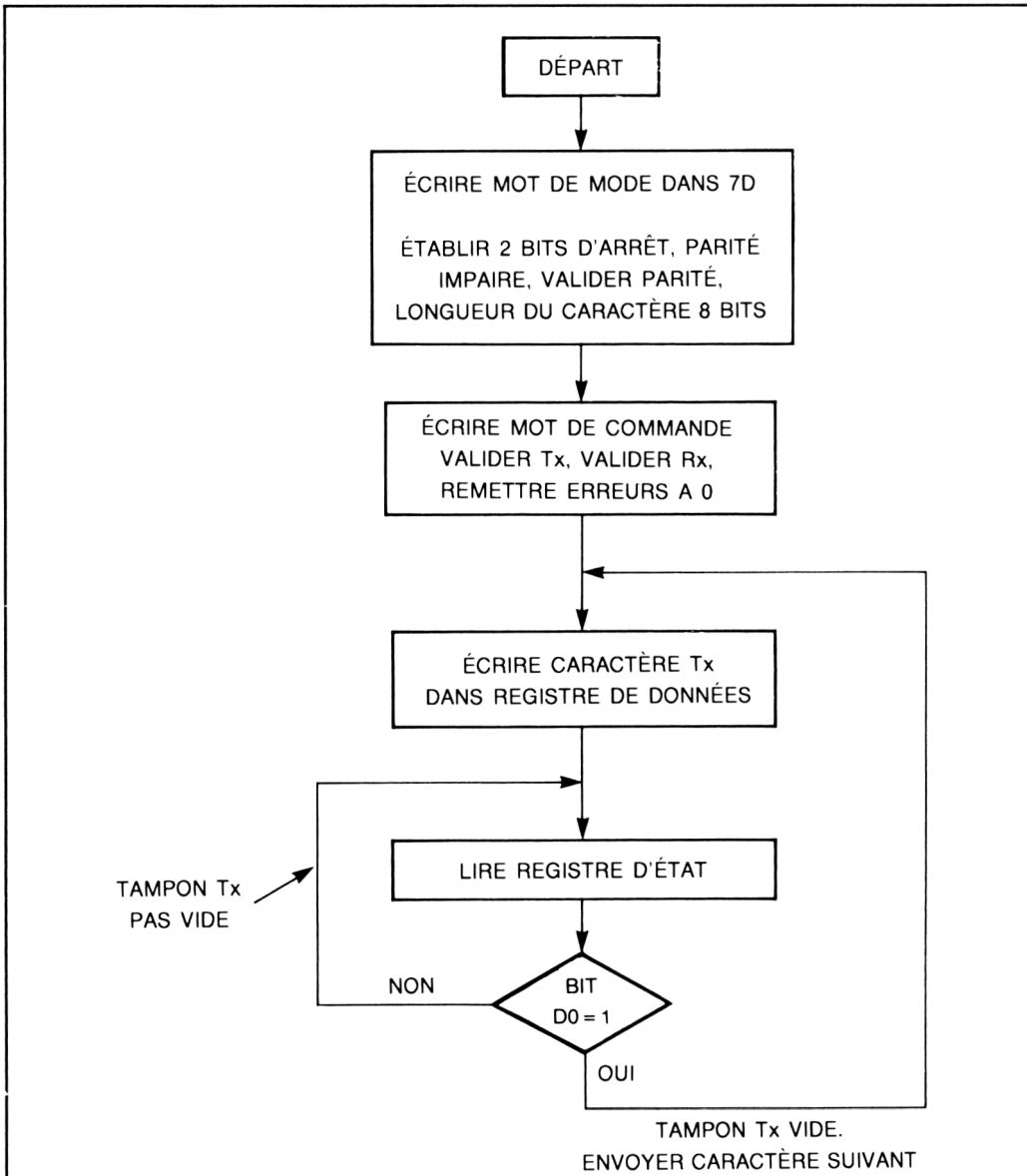


Figure 10.19 : *Ordinogramme de déroulement des opérations pour la transmission répétitive d'un caractère unique vers un autre dispositif série de réception.*

nexions des circuits. La Figure 10.22 donne l'ordinogramme de ce programme.

Les mnémoniques du Z80, qui permettent de réaliser l'ordinogramme de la Figure 10.22, sont donnés par la Figure 10.23.

```
1800    3EDD            LD A,0DDH       ; CHARGER MOT DE MODE DANS REGISTRE A
1802    D37D            OUT (7DH),A     ; SORTIE VERS 8251
;
; 2 BITS D'ARRET, PARITE IMPAIRE, VALIDER PARITE, CARACTERE DE 8 BITS
; FACTEUR X1 DU RYTHME DE BAUD
;
1804    3E15            LD A,15H        ; MOT DE COMMANDE DANS REGISTRE A
1806    D37D            OUT (7DH),A     ; SORTIE VERS 8251
;
; TX VALIDE, RX VALIDE, RAZ ERREURS
;
1808    3E49    LOOP1   LD A,49H        ;ASCII "I"
180A    D37C            OUT (7CH),A     ;SORTIE CARACTERE VERS TX
180C    DB7D    LOOP2   IN A,(7DH)      ;LIRE REGISTRE D'ETAT
180E    CB47            BIT 0,A         ;BIT DE TEST 0 = 1
1810    CA0C18          JP Z,LOOP2      ;TAMPON NON VIDE, CONTINUER SCRUTER
1813    C30818          JP LOOP1        ;TAMPON VIDE, CARACT. SUIVANT
;
```

Figure 10.20 : *Mnémoniques du Z80 pour réaliser l'ordinogramme de la Figure 10.19.*

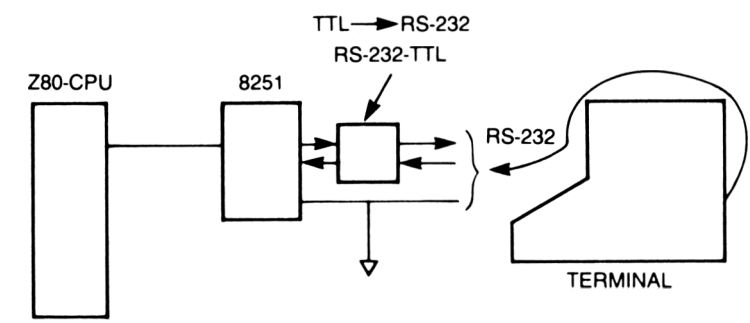


Figure 10.21 : *Schéma montrant comment se situe un USART dans un ensemble de communications entre un système à microprocesseur et un terminal.*

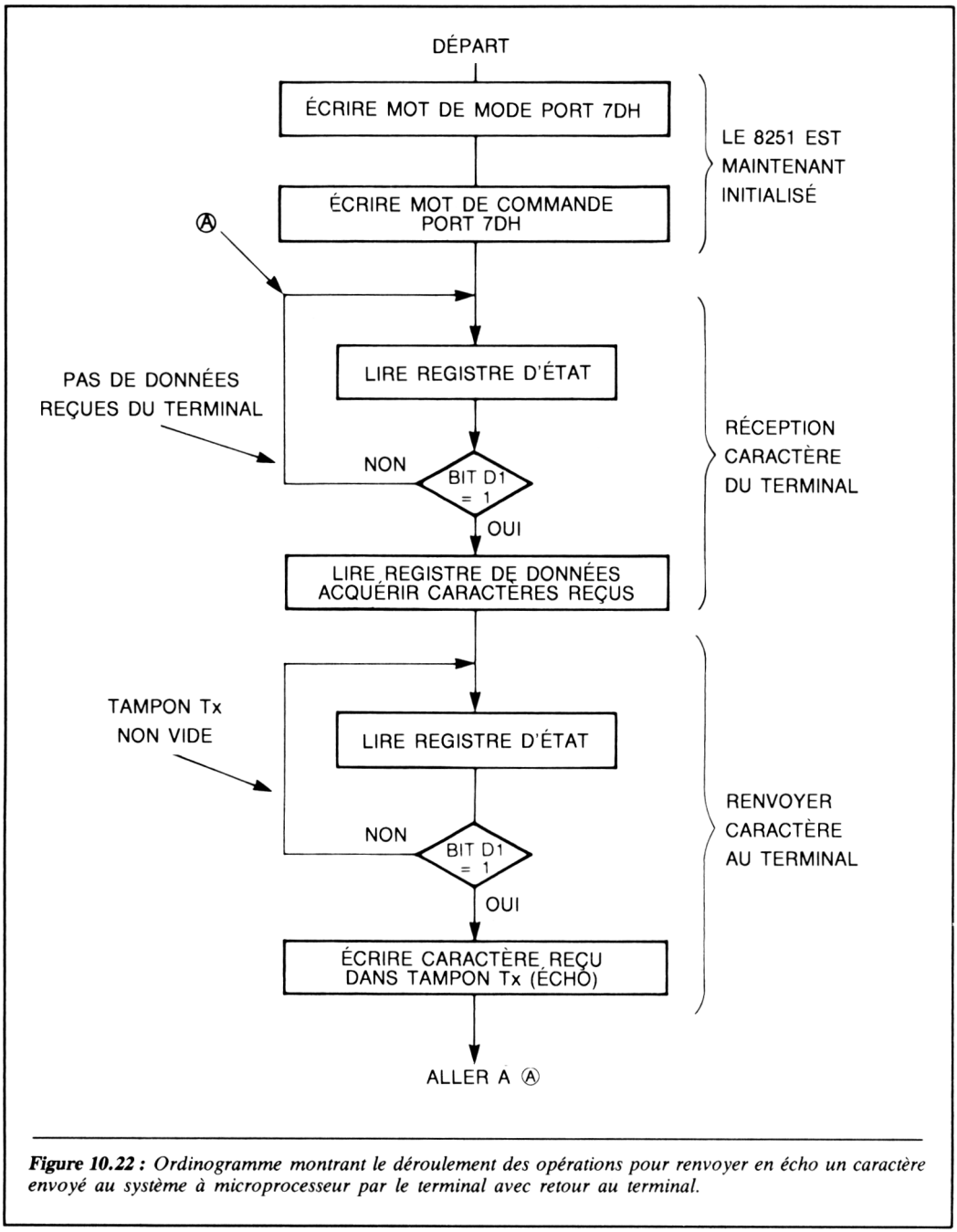


Figure 10.22 : *Ordinogramme montrant le déroulement des opérations pour renvoyer en écho un caractère envoyé au système à microprocesseur par le terminal avec retour au terminal.*

```
;
1800   3EDD            LD A,0DDH        ;MOT DE MODE
1802   D37D            OUT (7DH),A      ;ECRIRE DANS 8251
1804   3E15            LD A,15H         ;MOT DE COMMANDE
1806   D37D            OUT (7DH),A      ;ECRIRE DANS 8251
;
;  LE 8251 EST MAINTENANT INITIALISE
;
;  LE COMPOSANT ATTEND D'ABORD QU'UN CARACTERE
;  LUI SOIT ENVOYE PAR LE TERMINAL
;
1808   DB7D     LOOP1  IN A,(7DH)       ;LIRE REGISTRE D'ETAT
180A   CB4F             BIT 1,A          ;BIT DE TEST D1 = 1
180C   CA0818           JP Z,LOOP1       ;PAS PRET, CONTINUER A SCRUTER
;
;  ICI UN CARACTERE EST REÇU
;
180F   DB7C             IN A,(7CH)       ;LIRE LE CARACTERE
1811   47               LD B,A
;
;  ON NE CONTROLE PAS ERREURS DE DONNEES
;
;  MAINTENANT TRANSMISSION DES DONNEES
;
1812   DB7D     LOOP2  IN A,(7DH)       ;LIRE REGISTRE D'ETAT
1814   CB47            BIT 0,A          ;TEST D0 = 1
1816   CA1218           JP Z,LOOP2       ;XMIT PAS PRET, CONTINUER A SCRUTER
;
;  XMIT PRET A EMETTRE UN AUTRE CARACTERE
;
1819   78               LD A,B
181A   D37C             OUT (7CH),A      ;CARACTERE VERS 8251
181C   C30818           JP LOOP1         ;NOUVEAU DEPART
;
;  FIN DE LA ROUTINE ECHO
;
```

Figure 10.23 : *Mnémoniques du Z80 pour l'ordinogramme de la Figure 10.22.*

RESUME

Dans ce chapitre, nous avons examiné les bases de la communication série. Nous avons traité plusieurs points importants comportant vitesse de transfert, bits de départ, bits d'arrêt et bits de parité. Nous avons aussi analysé la signification de deux erreurs fondamentales que l'on rencontre parfois dans les communications série : les erreurs de cadrage et de dépassement.

Dans la seconde partie de ce chapitre, nous avons examiné un composant réel pour communication série : le 8251. Nous l'avons choisi parce qu'il est simple à utiliser et à comprendre et qu'il est d'un emploi très répandu dans l'industrie actuellement. En outre, il résout le problème de la communication série d'une façon simple et élégante.

Dans le Chapitre 11, nous analyserons le fonctionnement d'un autre composant pour communication série : le SIO du Z80. Il est légèrement plus compliqué que le 8251. En lisant ce chapitre, vous verrez que les renseignements donnés peuvent être aussi appliqués directement au boîtier SIO du Z80.

UTILISATION
DU Z80 - SIO

CHAPITRE 11

INTRODUCTION

Dans ce chapitre, nous allons étudier le dispositif d'entrée-sortie série appelé SIO. Il a de nombreux modes de fonctionnement différents et peut être employé dans beaucoup d'applications. Nous allons nous pencher sur l'utilisation la plus répandue : l'entrée-sortie asynchrone. Quand vous aurez compris comment est programmé le SIO et comment il est utilisé dans ce mode, vous pourrez comprendre plus facilement comment on l'emploie dans d'autres modes de fonctionnement. (*Note :* On supposera dans tout le chapitre que vous connaissez bien la théorie des communications asynchrones. Si ce n'est pas le cas, nous vous conseillons de vous reporter au Chapitre 10, sections 1 à 6.)

11.1 - Schéma du SIO

La Figure 11.1 donne le schéma du SIO. Examinons-le et portons notre attention sur les blocs principaux. Commençons par les deux blocs référencés canal A et canal B. Ils montrent que le SIO comporte deux canaux de communication série qui sont indépendants. Remarquez que chaque canal est associé à un bloc de commande et d'état. Ils sont reliés à plusieurs lignes d'entrée et de sortie. Nous étudierons celles-ci au fur et à mesure de ce chapitre.

Examinons maintenant les deux blocs situés à gauche des deux blocs de canaux. Ces blocs, référencés registres de lecture/écriture du canal A et du canal B, sont les logiques internes qui permettent la programmation appropriée du SIO.

Tous les modes de fonctionnement du SIO existent sous contrôle du logiciel. Avant d'employer le SIO, des mots de programmation doivent être envoyés à ces blocs pour « configurer » le composant. Nous analyserons la programmation de ces blocs plus loin dans ce chapitre.

Un autre bloc de la Figure 11.1 est la *logique de commande d'interruption*. Le SIO a une architecture d'interruption très puissante. Ce bloc logique permet au SIO de s'interfacer directement à la structure d'interruption du Z80, aussi bien qu'à d'autres dispositifs périphériques.

Le bloc référencé *logique de commande interne* sert à lire et à écrire des données dans le bus de données interne. Ce bloc synchronise de façon appropriée tous les transferts de données internes du SIO.

Enfin, le principal bloc d'entrée de données de la Figure 11.1, référencé *bus d'entrée-sortie du CPU*, sert d'adaptation et d'interface électrique entre CPU du Z80 et SIO. Nous apprendrons plus loin dans ce chapitre comment le SIO est connecté effectivement au microprocesseur Z80.

11.2 - Brochage du SIO

La Figure 11.2 montre le brochage du SIO. Enumérons chacune des broches.

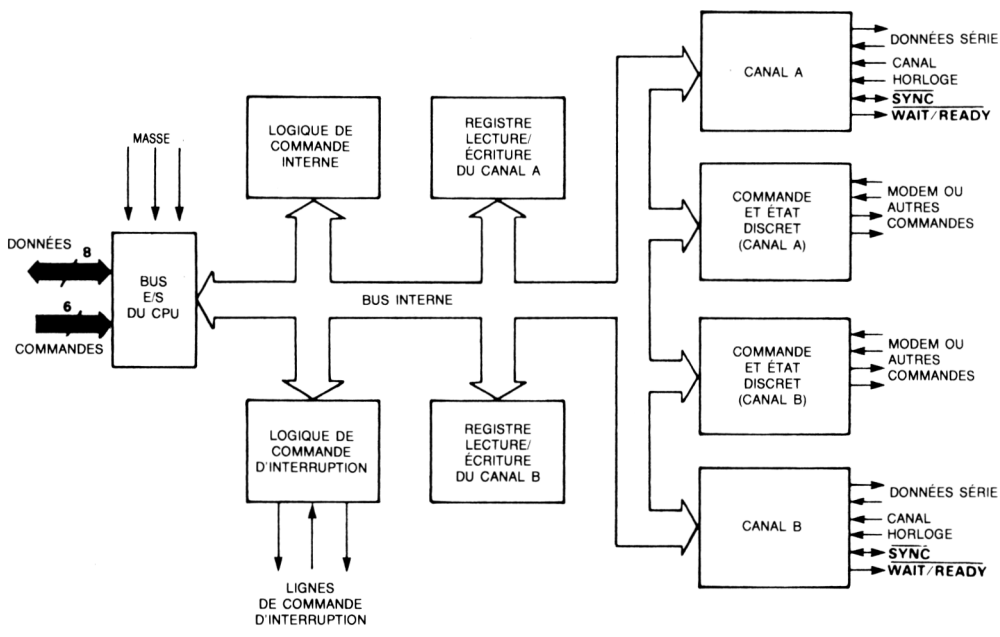


Figure 11.1 : *Schéma du Z80-SIO.*

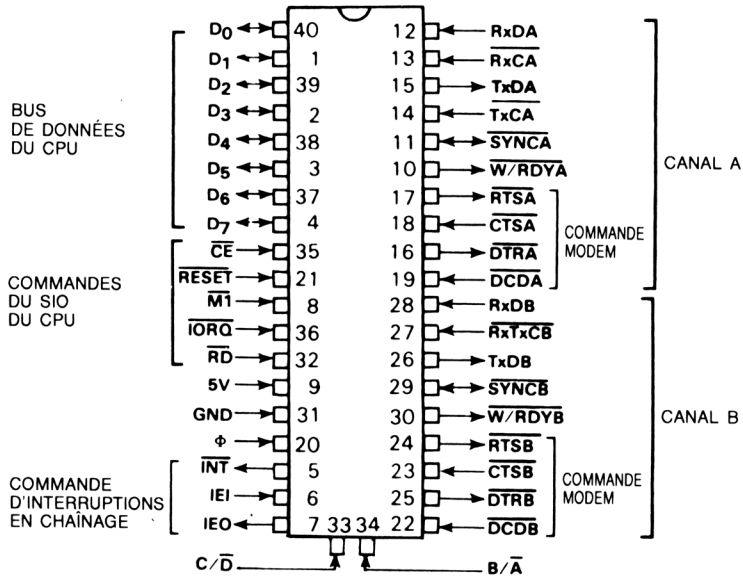


Figure 11.2 : *Brochage du Z80-SIO.*

D7-D0 Ce sont les lignes d'entrées et de sorties de données, reliées logiquement au bus de données du système à microprocesseur. Toutes les informations qui passent entre le SIO et le Z80 circulent par ces huit lignes de données. D7 est le bit de poids fort (MSB, *Most Significant Bit*).

B/$\overline{\text{A}}$ *(sélection du canal B ou A)* Chaque fois que cette ligne est au 1 logique, le canal B est sélectionné pour la communication ; chaque fois qu'elle est au 0 logique, c'est le canal A qui est choisi. Cette logique d'entrée est habituellement connectée à la ligne de sortie d'adresse A0 du système provenant du CPU Z80.

C/$\overline{\text{D}}$ *(ligne d'entrée de sélection de commande ou de données)* Chaque fois que cette ligne d'entrée est au 1 logique, le Z80 communique avec les registres de commandes du canal A ou B. Un 0 logique dans cette ligne d'entrée informe le SIO que les données en D0-D7 doivent être employées en données et non pas en information de commande du SIO.

Cette ligne d'entrée est habituellement reliée à la ligne d'adresse A1 du Z80.

Chaque fois que les lignes d'adresses A0 et A1 sont connectées aux broches B/$\overline{A}$ et C/$\overline{D}$, respectivement, il y a quatre ports distincts associés au dispositif physique. Ce sont les commandes ou les données des canaux A et B. (*Note :* Nous nous étendrons davantage sur l'adressage lorsque nous connecterons le SIO au Z80 dans la prochaine section de ce chapitre.)

$\overline{CE}$ C'est l'entrée *Chip Enable* (validation de boîtier), active au 0 logique. Cette entrée doit être active pour que le Z80 communique avec les registres internes de lecture ou d'écriture. Cette entrée provient généralement du décodage des six bits de poids fort de l'octet d'adresses de poids faible. Ce sont les lignes d'adresses A7-A2. Ces deux lignes d'adresses de poids faible vont accéder au port du dispositif.

CLOCK C'est l'entrée de l'horloge standard qui est également appliquée au CPU Z80, à la même fréquence ; elle va synchroniser les communications internes du SIO. L'entrée d'horloge a les mêmes caractéristiques, en continu, que celles du Z80 ; en d'autres termes, un 1 logique doit être égal à Vcc − 0,6 v. Remarquez que ce n'est pas une tension d'entrée en 1 logique TTL standard.

$\overline{M1}$ *(cycle machine 1)* Cette entrée est connectée à la ligne de sortie $\overline{M1}$ du Z80. Chaque fois que les entrées $\overline{M1}$ et $\overline{RD}$ du SIO sont toutes les deux au 0 logique, le Z80 recherche électriquement un code opération dans la mémoire du système. Le SIO est prévu pour reconnaître le code opération de l'instruction RETI. Chaque fois que les entrées $\overline{M1}$ et $\overline{IORQ}$ du SIO sont toutes les deux au 0 logique, le Z80 produit un accusé de réception d'interruption. Le SIO répond automatiquement à cette condition électrique, s'il est validé.

$\overline{IORQ}$ *(demande d'entrée et de sortie)* Cette ligne d'entrée est connectée à la ligne de sortie $\overline{IORQ}$ du microprocesseur Z80. Elle sert à informer électriquement le SIO que le CPU effectue une opération d'entrée ou de sortie. Elle indique aussi au SIO quand le Z80 accuse réception électriquement d'une demande d'interruption. Voir la description de $\overline{M1}$ pour les conditions de ce fonctionnement.

$\overline{RD}$ *(lecture)* Cette ligne d'entrée est connectée à la ligne de sortie $\overline{RD}$ du CPU Z80. Elle sert à informer le SIO chaque fois que le Z80 lit des données en mémoire ou en entrée-sortie. Chaque fois que $\overline{RD}$ et $\overline{M1}$ sont au 0 logique, le CPU recherche un code opération en mémoire. Si $\overline{RD}$, $\overline{IORQ}$ et $\overline{CE}$ sont au 0 logique, le CPU lit les données dans le SIO

en employant une opération d'entrée. Les données sont envoyées au CPU par le port sélectionné par les entrées C/$\overline{D}$ et B/$\overline{A}$.

Chaque fois que l'entrée $\overline{RD}$ est au 1 logique, et que $\overline{IORQ}$ et $\overline{CE}$ sont au 0 logique, le CPU écrit des données dans le SIO en employant une opération de sortie. Il n'y a pas d'entrée $\overline{WR}$ dans le SIO. Les données sont écrites dans le port sélectionné par les entrées C/$\overline{D}$ et B/$\overline{A}$.

$\overline{RESET}$ Un 0 logique à cette entrée inhibe les récepteurs et les transmetteurs du canal A et du canal B. Les sorties des transmetteurs des deux canaux vont à l'état *marquage*. Toutes les commandes de modem sont mises au 1 logique. Toutes les interruptions sont inhibées. Le SIO doit être mis à l'état initial (*reset*) par commande logicielle et tous les registres internes doivent être reprogrammés après *reset* par circuit extérieur.

IEI *(entrée de validation d'interruption)* Chaque fois que cette ligne d'entrée est au 1 logique actif, le système d'interruption hardware du SIO est validé. La ligne IEI est habituellement reliée à la ligne IEO d'un autre périphérique pour être utilisée dans un chaînage de priorité d'interruption. Pour permettre les interruptions au SIO, l'entrée doit être au 1 logique. Pour que le SIO réponde aux interruptions, le logiciel doit aussi le valider. Nous analyserons les interruptions en détail, plus loin dans ce chapitre.

IEO *(sortie de validation d'interruption)* Cette ligne de sortie est active au 1 logique. Elle sert à informer électriquement les périphériques connectés au chaînage d'interruption qu'aucun élément ayant priorité inférieure ne peut causer une interruption. Si le SIO n'est pas employé dans un chaînage d'interruption, cette ligne de sortie peut être ignorée électriquement.

$\overline{INT}$ *(sortie de demande d'interruption)* Cette sortie est à drain ouvert. Ceci lui permet d'être connectée directement à la broche d'entrée $\overline{INT}$ du Z80. Chaque fois que le SIO demande une interruption, la sortie est au 0 logique. Dans les autres cas, la sortie est ouverte. Une résistance de rappel est habituellement connectée à la broche d'entrée $\overline{INT}$ du CPU pour ramener tous les systèmes à drain ouvert au 1 logique chaque fois qu'ils ne sont pas actifs.

W/$\overline{RDYA}$, W/$\overline{RDYB}$ Ce sont les lignes WAIT/RDY qui sont associées aux canaux A et B. Ces sorties sont définies par programme. Si elles sont définies en fonction attente (*wait*), elles sont à drain ouvert. Si elles sont définies en fonction prêt (*ready*), elles sont amenées en état actif haut et bas. Une fonction prêt est employée avec des contrôleurs de DMA pour indiquer si le SIO a des données à envoyer ou s'il est prêt à en accepter.

La fonction attente est utilisée chaque fois que le Z80 communique avec le SIO pour indiquer quand celui-ci a des données prêtes pour le CPU ou bien quand il a de la place pour que le CPU écrive des données.

$\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$ *(mettre à zéro pour envoyer A, B)* Ces signaux d'entrée sont actifs à l'état bas. Ils sont valides sous contrôle software en *auto-validation*. Dans ce mode ils permettent au circuit extérieur de démarrer l'opération de transmission. Si ces signaux d'entrée ne sont pas programmés en mode autó-validation, ils peuvent être utilisés comme entrées générales du SIO.

$\overline{\text{DCDA}}$, $\overline{\text{DCDB}}$ *(détection de porteur de données A, B)* Ces signaux d'entrée sont actifs au niveau bas. Lorsqu'ils sont actifs, ils valident le récepteur A ou B.

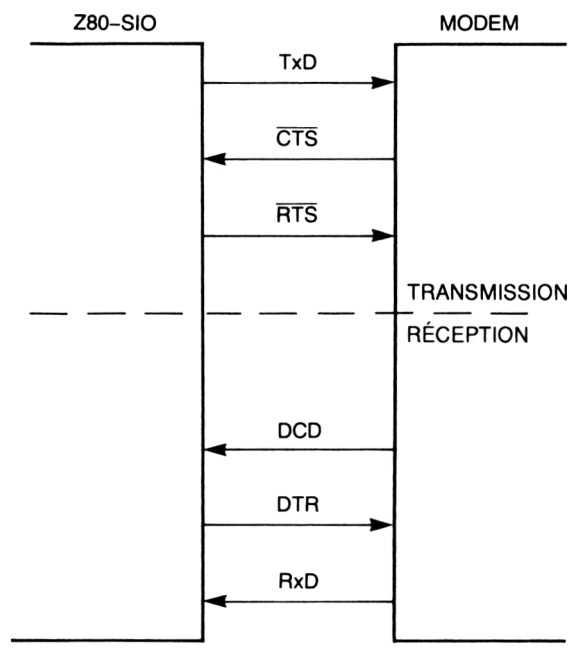


Figure 11.3 : *Schéma montrant les principaux signaux de commandes de modem du Z80-SIO.*

$\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$ *(demande d'envoi A, B)* Ces signaux de sortie sont actifs au niveau bas. Le bit RTS peut être établi par programme. La sortie va au 1 logique quand le transmetteur est vide.

$\overline{\text{DTRA}}$, $\overline{\text{DTRB}}$ *(terminal de données prêt A, B)* Ces signaux de sortie suivent l'état logique du bit programmé.

(*Note :* Les quatre signaux ci-dessus $\overline{\text{CTS}}$, $\overline{\text{DCD}}$, $\overline{\text{RTS}}$ et $\overline{\text{DTR}}$ peuvent servir à la commande d'un modem. La Figure 11.3 donne un schéma montrant les liaisons de ces signaux à un modem.)

RxDA, RxDB *(entrée de réception de données)* Ces entrées sont non inversées quand les données sont reçues.

TxDA, TxDB *(sortie de transmission de données)* Ces sorties sont non inversées quand les données sont transmises.

$\overline{\text{RxCA}}$, $\overline{\text{RxCB}}$ Ce sont les broches d'entrée d'horloge de récepteur du boîtier. Les horloges peuvent être 1, 16, 32 ou 64 fois le rythme de données du récepteur.

$\overline{\text{TxCA}}$, $\overline{\text{TxCB}}$ Ces deux entrées sont les broches d'entrée d'horloge du transmetteur. Les horloges peuvent être 1, 16, 32 ou 64 fois le rythme de données de transmission en fonctionnement asynchrone. Le facteur de multiplication du canal récepteur et du canal transmetteur doit être le même, mais les rythmes des données du transmetteur et du récepteur peuvent être différents.

Par exemple, le rythme de données du transmetteur peut être 2400 bauds et celui du récepteur 1200 bauds. Les facteurs de multiplication des deux blocs doivent être égaux. Si le multiplicateur est mis à 16, la fréquence d'entrée d'horloge du transmetteur est 2400×16 hertz ; la fréquence d'horloge du récepteur est 1200×16 hertz. Ces deux fréquences sont différentes, mais les facteurs de multiplication sont les mêmes.

11.3 - Connexion du SIO aux bus du Z80

La Figure 11.4 montre la connexion type du SIO au microprocesseur Z80. On suppose dans ce schéma, que la charge en continu du bus de données ne nécessite pas de tampons de données, c'est pour cela qu'il n'en figure pas.

La broche d'entrée 35 $\overline{\text{CE}}$ est décodée à partir des lignes d'adresses A7-A2. On utilise un 74LS138 pour le décodage. Quand les lignes d'adresses

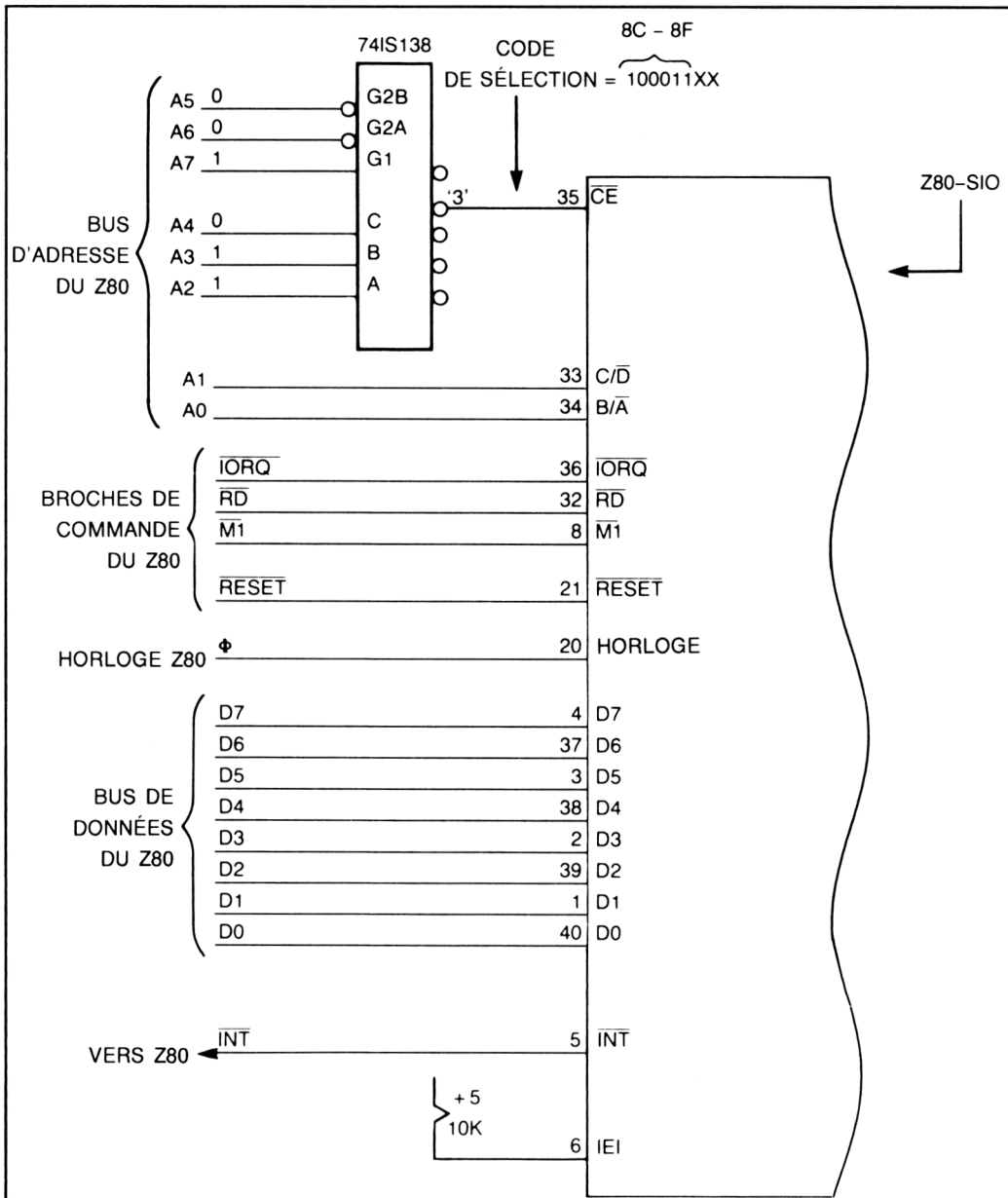


Figure 11.4 : *Schéma de toutes les liaisons entre Z80-CPU et Z80-SIO. Cette application ne comporte pas d'amplificateur de bus de données.*

A7-A2 sont à 100011XX, le boîtier est validé. XX indique que les conditions de A1 et A0 sont *sans importance* pour que $\overline{CE}$ devienne actif. A1 et A0 sont connectés aux entrées C/$\overline{D}$ et B/$\overline{A}$, respectivement.

Avec ces connexions électriques et ces décodages, les adresses de port du SIO sont 8C, 8D, 8E et 8F. Ces ports sont définis comme suit :

```
8C = Données A
8D = Données B
8E = Commande A
8F = Commande B
```

La commande $\overline{INT}$ du SIO est connectée à la broche d'entrée $\overline{INT}$ du Z80. IEI est ramené au + 5 volts par une résistance de 10 KΩ, ceci pour valider le circuit d'interruption du dispositif. IEO n'est pas employé dans cette application. La Figure 11.5 montre comment IEI et IEO seraient connectés si le SIO était utilisé dans un ensemble à interruption prioritaire avec d'autres périphériques.

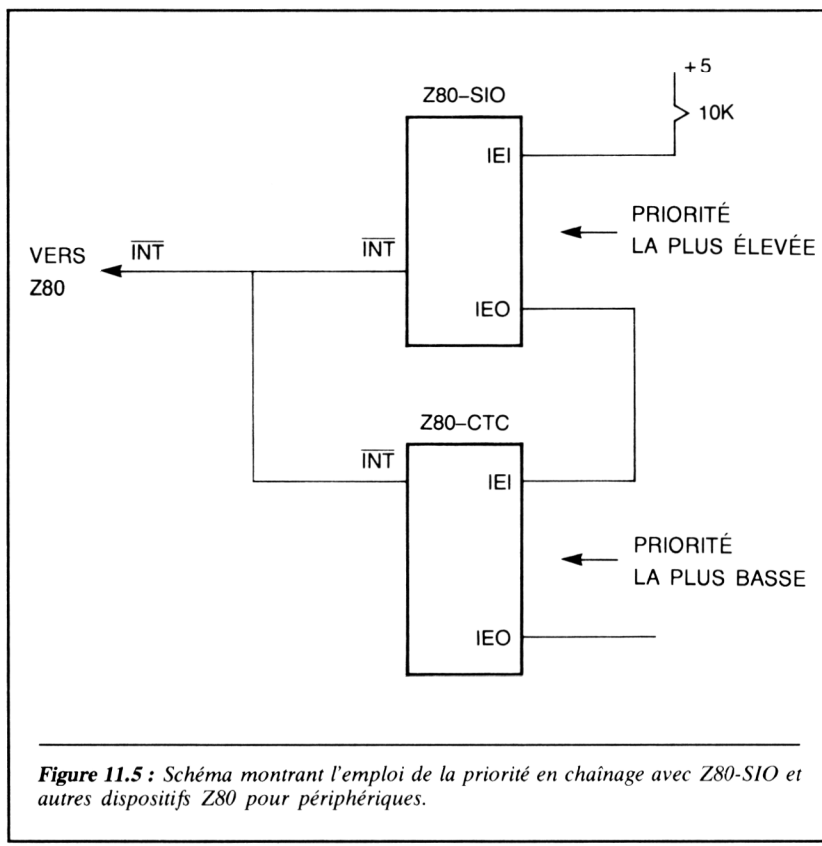


Figure 11.5 : *Schéma montrant l'emploi de la priorité en chaînage avec Z80-SIO et autres dispositifs Z80 pour périphériques.*

11.4 - Connexion du SIO aux lignes de transmission série

Pour effectuer cette connexion, on supposera que le SIO utilise les canaux A et B dans un plan de communication asynchrone série. La liaison série s'effectue par RS-232. La Figure 11.5 montre les liaisons permettant l'emploi du SIO dans ce mode. Remarquez dans la Figure 11.6 que, pour les deux canaux, seuls TxD, RxD, $\overline{\text{TxC}}$ et $\overline{\text{RxC}}$ ont besoin d'être connectés.

11.5 - Registres du SIO

Une fois le circuit connecté comme décrit dans la section précédente, la tâche à effectuer consiste à programmer le SIO pour le faire communiquer

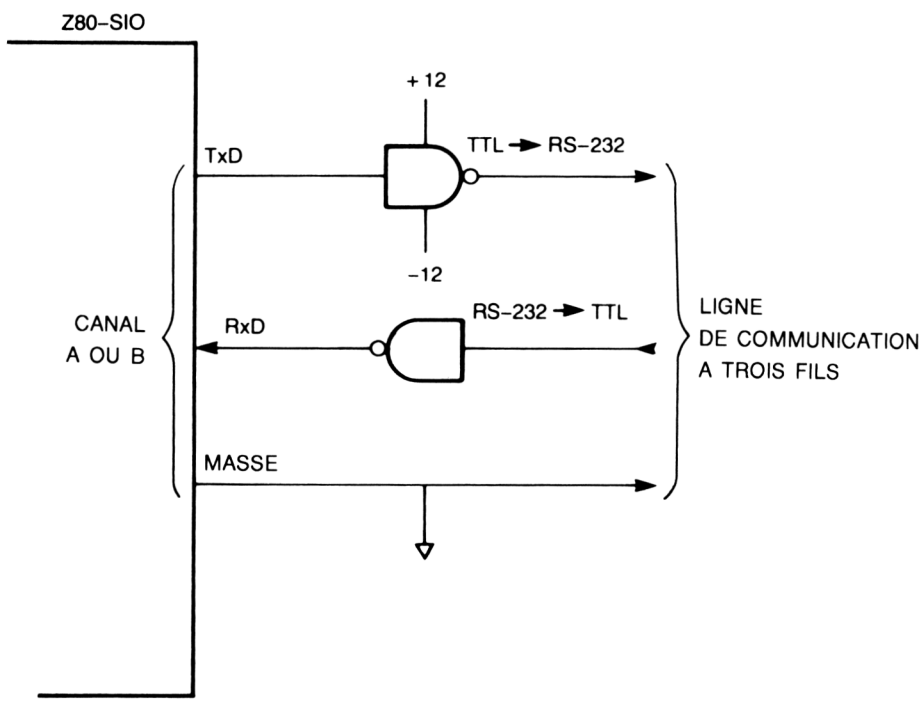


Figure 11.6 : *Schéma montrant l'emploi d'adaptateur de tension entre niveau TTL du SIO et niveau du bus RS-232.*

conformément aux nécessités de votre application. Dans les sections suivantes de ce chapitre, nous utiliserons les registres du SIO, et donnerons des exemples de programmes permettant d'initialiser le SIO. Ceci devrait vous aider à le programmer.

Le SIO contient huit registres internes d'écriture pour le canal B, c'est-à-dire WR0-WR7 et sept registres d'écriture pour le canal A, c'est-à-dire WR0-WR1 et WR3-WR7 (sans WR2). Le registre d'écriture WR2 contient le vecteur d'interruption des deux canaux (A et B). Le bit de poids fort de chaque registre est D7.

Il faut deux octets pour écrire des données dans n'importe quel registre excepté WR0. Le premier octet est écrit dans WR0. Il est décodé d'une façon interne et il pointe le registre pour l'octet suivant. Ceci est nécessaire car il n'y a pas de lignes d'adresses pour pointer les registres internes. Toutes les commandes fondamentales peuvent être écrites dans WR0 par un seul octet. Vous comprendrez mieux à mesure que nous donnerons des exemples.

Le canal B a un jeu de trois registres de lecture (référencés RR0-RR2) qui contiennent l'état du SIO. Le canal A n'a pas de registre RR1 comme vous l'apprendrez au cours de ce chapitre.

Après cette brève présentation, vous pouvez consulter l'Annexe et examiner chacun des registres d'écriture et de lecture du SIO. Cette annexe donne des renseignements directement extraits du Manuel de données Mostek et décrit en détail les registres internes du SIO. Ceci est important pour l'apprentissage de la programmation du SIO. Une fois familiarisé avec ce matériel, continuez à lire l'explication d'une séquence de programmation type, afin que vous voyiez exactement comment on peut utiliser ce composant.

11.6 - Séquence générale d'initialisation du SIO

Nous allons maintenant montrer le logiciel nécessaire au fonctionnement du SIO en scrutation. Notre analyse va porter sur différents aspects du fonctionnement du SIO. D'une façon générale, vous allez apprendre comment programmer le dispositif et l'interfacer au SIO pour fonctionnement série à scrutation.

Examinons certaines caractéristiques générales du SIO. Nous allons commencer par examiner certains points généraux concernant les registres (expliqués en détail dans l'Annexe). En premier lieu, le SIO peut recevoir ou transmettre 5, 6, 7 ou 8 bits par caractère. Le nombre de bits est sous contrôle du programme et peut être différent pour le transmetteur et pour le récepteur.

En transmission asynchrone, le SIO ajoutera automatiquement un bit de départ - 0 logique - au début du caractère. Il ajoute aussi 1, 1 1/2 ou 2 bits d'arrêt à la fin de chaque transmission de caractère. Le nombre de bits d'arrêt est sous contrôle du programme. Le bit d'arrêt est au 1 logique, ce qui est le niveau logique de l'état de marquage du SIO.

Avant d'ajouter des bits d'arrêt au caractère, on peut ajouter un bit de parité. (*Note :* La notion de parité a été expliquée au Chapitre 10, section 10.3.) Une fois la parité choisie et programmée, le SIO fonctionne automatiquement avec parité paire ou impaire, au choix. Si la parité n'a pas été choisie, le SIO n'ajoute *pas* de bit supplémentaire pour établir une parité paire ou impaire.

Quand les données provenant du SIO sont lues par le CPU, huit bits sont entrés. Le caractère reçu n'est pas nécessairement constitué par tous ces bits. Si, par exemple, il n'y a que six bits, les six premiers, D0-D5, constitueront alors le caractère, D0 étant le bit de poids faible. Les autres bits du flot de données en série seront les bits d'arrêt et, éventuellement, le bit de parité. Un bit d'arrêt est toujours un 1 logique. S'il y a des bits inutilisés, ils sont mis au 1 logique ou au niveau de marquage.

Supposons, par exemple, que le caractère reçu soit spécifié comme ayant cinq bits, sans parité et avec un bit d'arrêt. Cela signifie que le caractère reçu a seulement 5 bits + 1 bit d'arrêt. Les deux autres bits sont constitués par le niveau de marquage de SIO ou par des 1 logiques.

Il y a quatre entrées d'horloge pour communication série et une entrée CLK *(CLocK)* dans le SIO. Cette entrée CLK distincte sert à synchroniser les transferts de données internes et n'est donc pas impliquée dans la transmission série. Les quatre horloges utilisées sont RxA, RxB, TxA et TxB. La fréquence d'horloge d'entrée de ces entrées peut être 1, 16, 32 ou 64 fois le rythme de transmission ou de réception des données.

Maintenant que nous connaissons le principe du SIO, continuons à voir notre exemple de communication asynchrone à scrutation. Ce qui suit est une séquence (ou initialisation de registre), qui doit être suivie quand on emploie le SIO en mode de fonctionnement asynchrone.

1. *Ecrire dans WR0.* Cette action initialise le SIO. L'initialisation complète du SIO peut nécessiter l'écriture de plusieurs octets. Ces octets peuvent être nécessaires pour envoyer certains des octets de commandes spéciaux décrits dans la section du WR0.

2. *Ecrire dans WR1.* Cette action établit le diviseur des signaux d'horloge de réception et de transmission, les bits d'arrêt et de parité.

3. *Ecrire dans WR3.* Cette action établit le nombre de bits de réception, l'auto-validation et la validation du récepteur.

4. *Ecrire dans WR5.* Cette action spécifie le nombre de bits Tx et met le bit de validation Tx.

5. *Ecrire dans WR2.* Pour le canal B seulement. Le vecteur d'arrêt d'interruption est spécifié. Une écriture dans WR2 ne doit être effectuée que pour un système commandé par des interruptions.

6. *Ecrire dans WR1.* Cette action spécifie le système d'interruption du SIO. De plus elle valide ou inhibe la structure d'interruption du SIO.

Les registres WR6 et WR7 ne sont pas employés dans le mode de communication asynchrone. Ils sont nécessaires pour les communications synchrones.

Pour montrer comment fonctionne cette initialisation dans un système type comportant un SIO, donnons un exemple. Nous allons établir le dispositif pour le canal A du SIO. Les paramètres de communication série vont être :

- 64 = facteur de multiplication des signaux d'horloge
- 2 bits d'arrêt
- Caractères de 8 données
- Interruptions inhibées
- Pas de parité

Un programme Z80 initialisant un SIO comme ci-dessus se trouve en Figure 11.7. On suppose ici que le SIO se trouve aux emplacements d'entrée-sortie 8CH, 8DH, 8EH et 8FH.

11.7 - Réception de données série

Maintenant que le SIO est initialisé, nous allons commencer à recevoir des données du SIO avec le CPU. Dans le fonctionnement en mode de scrutation, on suppose que le Z80 va saisir les caractères provenant du SIO aussi vite qu'ils sont reçus de la ligne d'entrée série. Si ce n'était pas le cas, il faudrait une procédure de *handshaking* sur la ligne série pour garantir que les caractères ne soient pas reçus à une vitesse dépassant celle nécessitée par le CPU pour leur traitement. Si c'est le cas, on dit que le récepteur a une erreur de surcharge.

Le SIO a une mémoire incorporée FIFO (*First-In, First-Out,* c'est-à-dire premier entré, premier sorti) triple. Donc, trois mots peuvent être reçus avant que ne se pose un problème de surcharge. Les données reçues sont appliquées au SIO huit bits à la fois, donnant ainsi au CPU le temps d'entrer le caractère et de le stocker en mémoire avant que le caractère suivant ne soit complètement reçu par le SIO.

```
;
;   VOICI UN PROGRAMME Z80 POUR INITIALISATION DU SIO
;   HORLOGE X64, 2 BITS D'ARRET, 8 BITS RX, 8 BITS TX, PAS D'INT.
;
;
;
;   ETABLIR EQU DE PORT E/S
;
008C            ADATA   EQU   8CH    ;PORT DE DONNEES CANAL A DU SIO
008D            BDATA   EQU   8DH    ;PORT DE DONNEES CANAL B DU SIO
008E            ACON    EQU   8EH    ;PORT DE COMMANDE CANAL A DU SIO
008F            BCON    EQU   8FH    ;PORT DE COMMANDE CANAL B DU SIO
;
;
1800   0E8E             LD C,ACON   ;CHARGER ADRESSES E/S DANS REG. C
1802   3E18             LD A,18H
1804   ED79             OUT (C),A   ;WR0, RAZ CANAL
1806   3E04             LD A,04H    ;CHARGER A AVEC POINTEUR WR4
1808   ED79             OUT (C),A   ;POINTER WR4
180A   3ECC             LD A,0CCH
180C   ED79             OUT (C),A   ;WR4, HORL. X64, 2 BITS D'ARRET, PAS D'INT.
180E   3E03             LD A,03H    ;CHARGER A AVEC POINTEUR WR3
1810   ED79             OUT (C),A   ;POINTER WR3
1812   3EC1             LD A,0C1H   ;8 BITS DE RECEPTION, VALIDATION RECEPT.
1814   ED79             OUT (C),A
1816   3E05             LD A,05H    ;CHARGER A AVEC POINTEUR WR5
1818   ED79             OUT (C),A   ;POINTER WR5
181A   3E68             LD A,68H
181C   ED79             OUT (C),A   ;8 BITS TX, VALIDATION TX
181E   3E01             LD A,01H    ;CHARGER A AVEC POINTEUR WR1
1820   ED79             OUT (C),A   ;POINTER WR1
1822   3E00             LD A,00H
1824   ED79             OUT (C),A   ;INHIBER TOUTES INTERRUPTIONS
;
;
;   APRES EXECUTION DE CE CODE LE SIO EST INITIALISE
;   ET PRET A COMMUNIQUER
;
                        END
```

Figure 11.7 : *Programme Z80 pour initialiser le Z80-SIO.*

Dans la routine de réception, nous allons vérifier après chaque caractère qu'il n'y a pas d'erreur de surcharge. Remarquez que si c'est le cas, le CPU doit remettre à zéro le drapeau d'erreur du SIO avant qu'il ne puisse vérifier le caractère suivant. Les drapeaux d'erreur restent positionnés jusqu'à ce qu'ils soient mis à zéro. Un ordinogramme de réception de caractère est montré en Figure 11.8.

Comme on peut le voir dans cette figure, la première opération à effectuer est la lecture du registre d'état RR0. Cette opération a pour but de permettre de voir si le tampon de réception est plein. S'il y a seulement un caractère dans le FIFO du récepteur, le bit D0 de RR0 est mis au 1 logique. Si le bit D0 est au 0 logique, le CPU va reboucler continuellement dans l'attente d'un caractère ou bien il va traiter d'autres informations et revenir plus tard scruter ce registre. Dans la Figure 11.8, le CPU reboucle sans arrêt, scrutant le registre jusqu'à ce qu'un caractère ait été reçu.

Après réception d'un caractère par le SIO et une fois le bit D0 de RR0 au 1 logique, le caractère est lu. Ensuite RR1 est lu pour voir si le caractère comporte des erreurs. Les bits D6, D5 et D4 de RR1 indiquent s'il y a des erreurs. D6 indique une erreur de cadrage, D5 une erreur de surcharge et D4 une erreur de parité. Si l'on trouve des erreurs, le SIO doit alors avoir des erreurs remises à zéro en écrivant dans WR0.

Si des erreurs sont détectées, le logiciel doit reporter le type d'erreur à la console du système (ou bien prendre toute mesure s'imposant dans votre application).

La Figure 11.9 montre un programme permettant de réaliser l'ordinogramme de réception de caractère en mode de scrutation.

11.8 - Transmission de caractère en mode scruté

Dans la section précédente, nous avons présenté un programme permettant de recevoir un caractère en mode de fonctionnement à scrutation. Nous allons maintenant donner un programme de transmission de caractère en mode de scrutation. La Figure 11.10 donne l'ordinogramme correspondant.

La première opération y figurant est une lecture de RR0. Souvenez-vous que RR0 peut être lu sans écriture préalable d'un mot dans le registre WR0 pour pointer. Le bit D2 de RR0 est ensuite testé.

Si le bit D2 est au 1 logique, le tampon de transmission est vide. Si le bit D2 est au 0 logique, le tampon de transmission est plein. L'ordinogramme doit comporter une boucle d'attente jusqu'à ce que le tampon Tx soit vide. Quand il est vide, le caractère de transmission est écrit dans le registre de données du canal et le SIO démarre automatiquement la séquence de transmission. La Figure 11.11 montre un programme Z80 réalisant l'ordinogramme de la Figure 11.10.

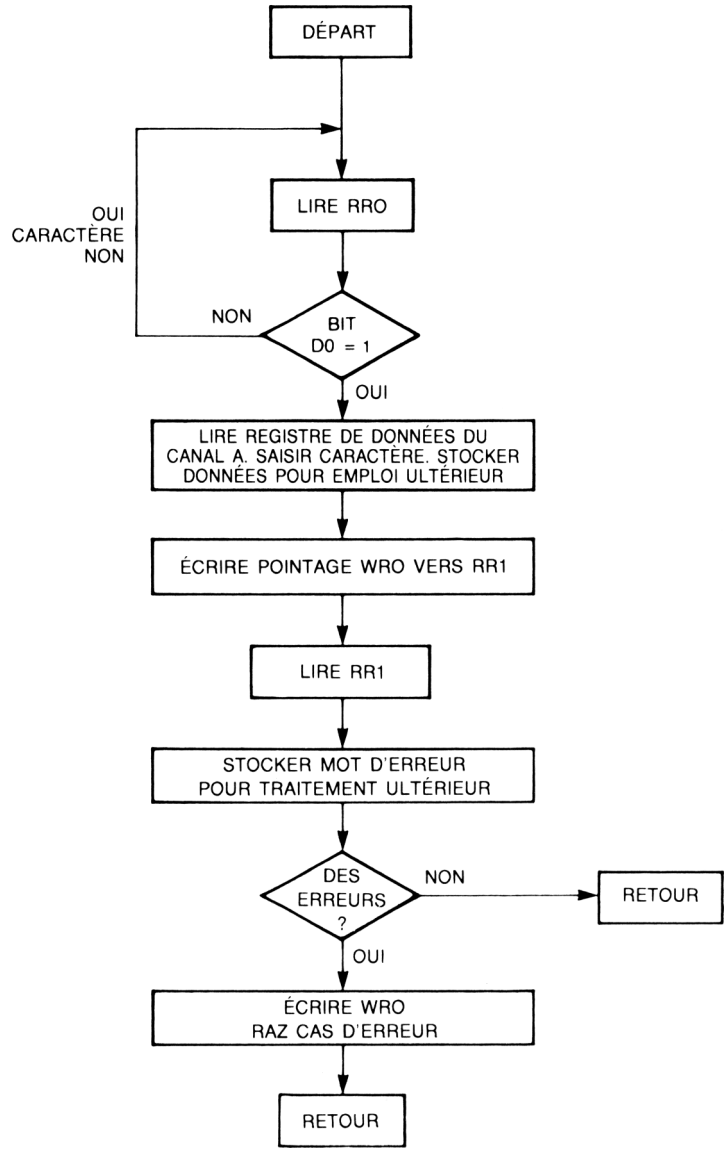


Figure 11.8 : *Ordinogramme montrant le déroulement des opérations nécessaires pour lire un caractère dans le Z80-SIO.*

```
;
;   CETTE ROUTINE VA RECEVOIR UN CARACTERE
;   EN MODE DE SCRUTATION. ELLE RETOURNE
;   APRES RECEPTION D'UN CARACTERE.
;   LE CARACTERE REVIENT DANS UN EMPLACEMENT
;   MEMOIRE « CHIN » ET LES ERREURS DANS UN EMPLACEMENT
;   MEMOIRE « CHER ». SI LES BITS DE CHER
;   D6, D5, D4 = 0, IL N'Y A PAS D'ERREUR.
;
;   ON SUPPOSE L'ADRESSAGE SIO AUX ADRESSES DE PORT 8C-8F,
;   SIO INITIALISE
;
;
;
008C            ADATA   EQU  8CH       ;PORT DE DONNEES CANAL A DU SIO
008E            ACON    EQU  8EH       ;PORT DE COMMANDE CANAL A DU SIO
5900            CHIN    EQU  5900H     ;EMPLACEMENT MEMOIRE POUR ENTREE CARACTERE
5901            CHER    EQU  CHIN+1    ;EMPLACEMENT MEMOIRE POUR ERREURS REÇUES
;
;
2400   DB8E     LOOP    IN A,(ACON)    ;'LIRE RRO
2402   E601             AND 01H        ;VERIFIER DO POUR PRET
2404   CA0024           JP Z, LOOP     ;PAS PRET CONTINUER A SCRUTER
;
;   MAINTENANT LECTURE CARACTERE DANS CANAL A
;
2407   DB8C             IN A,(ADATA)   ;LIRE CARACTERE
2409   320159           LD (CHER),A    ;LE STOCKER
;
;  MAINTENANT ON VERIFIE ERREURS REÇUES
;
240C   3E01             LD A,01H
240E   D38E             OUT (ACON),A   ;ETABLIR POINT. WRO VERS RR1
2410   DB8E             IN A,(ACON)    ;LIRE REG. D'ERREUR RR1
2412   320159           LD (CHER),A    ;STOCKER ERREURS
2415   E670             AND 70H        ;MASQUER TOUT SAUF D6, D5, D4
2417   C8               RET Z          ;SI 0 ALORS PAS D'ERREURS
;
;   SI PAS D'ERREURS, RETOUR DE ROUTINE
;   SI ERREURS, RAZ AVANT RETOUR
;
2418   3E30             LD A,30H       ;ETABLIR COMMANDE DE RAZ D'ERREUR
241A   D38E             OUT (ACON),A   ;RAZ COMMANDE D'ERREUR WRO
241C   C9               RET
;
;
;
                        END
```

Figure 11.9 : *Programme Z80 réalisant l'ordinogramme de la Figure 11.8.*

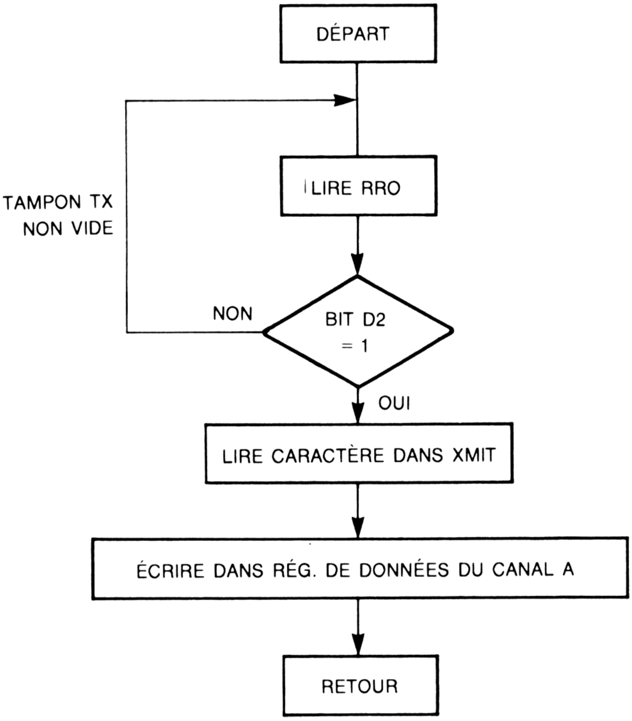


Figure 11.10 : *Ordinogramme montrant le déroulement des opérations pour transmettre un caractère au moyen du Z80-SIO.*

```
;
;   CETTE ROUTINE SERT A TRANSMETTRE
;   UN CARACTERE AVEC SIO
;   EN MODE DE SCRUTATION
;
;   CARACTERE A TRANSMETTRE STOCKE EN « CHTX ».
;   ON SUPPOSE SIO INITIALISE
;   ET ADRESSE EN
;   8C-8F.
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Figure 11.11 : *Mnémoniques Z80 réalisant l'ordinogramme de la Figure 11.10. (...)*

```
;
;
008C            ADATA   EQU 8CH         ;PORT DE DONNEES DU CANAL A DU SIO
008E            ACON    EQU 8EH         ;PORT DE COMMANDE DU CANAL A DU SIO
5902            CHTX    EQU 5902H       ;EMPLACEMENT MEMOIRE DU CARACTERE
;
;
2500  DB8E      LOOP1   IN A,(ACON)     ;LIRE VERIF. RRO POUR TX VIDE
2502  CB57              BIT 2,A         ;TESTER BIT D2
2504  CA0025            JP Z,LOOP1      ;D2 = 1 SI VIDE, 0 AUTREMENT
;
;  TAMPON TX MAINTENANT VIDE
;  LIRE CARACTERE A TRANSMETTRE
;
2507  3A0259            LD A,(CHTX)     ;CHARGER CARACTERE DANS REG. A
250A  D38C              OUT (ADATA),A   ;SORTIR CARACTERE VERS TX
;
;  CARACTERE MAINTENANT TRANSMIS
;  RETOUR DE ROUTINE
;
250C  C9                RET
                        END
```

Figure 11.11 : *Mnémoniques Z80 réalisant l'ordinogramme de la Figure 11.10 (Suite).*

11.9 - Interruptions avec le SIO

Maintenant que nous savons comment les caractères sont transmis et reçus en mode de fonctionnement à scrutation, apprenons comment on peut utiliser le SIO avec des interruptions. Nous allons commencer par revoir la structure d'interruption du SIO. Nous allons donner des exemples d'initialisation avec SIO, puis transmettre et recevoir des routines en mode interruption. Vers la fin de cet exposé, vous devriez avoir une assez bonne idée de la façon d'utiliser le SIO avec des interruptions.

Dans le registre du SIO, WR2 sert à stocker le vecteur d'interruption. Ce vecteur est placé dans le bus de données en réponse à un accusé de réception d'interruption du Z80. On suppose que le Z80 fonctionne en mode 2 d'interruption. Seul le canal B du SIO a le registre de vecteur d'interruption. Cela n'exclut cependant pas l'emploi du canal A en mode d'interruption.

Il y a huit vecteurs d'interruption différents que le SIO peut placer automatiquement dans le bus de données du système. Ces huit vecteurs sont formés à partir du vecteur écrit pendant la phase d'initialisation du SIO. Pour permettre la formation de ces huits vecteurs, le bit D2 du registre WR1, appelé *vecteur d'affectation d'état,* doit être mis au 1 logique.

Si ce bit est mis à 1 et que les vecteurs d'interruption sont valides, le type d'interruption va alors modifier certains bits du vecteur d'interruption dans WR2. Les bits du vecteur qui sont modifiés sont D3, D2 et D1. En modifiant trois bits, il y a huit différents vecteurs possibles.

Les huit vecteurs différents sont divisés en deux groupes de quatre. Chaque groupe est employé par un canal différent. On peut donc avec un vecteur obtenir l'information pour le canal A aussi bien que pour le canal B. A ce moment-là se pose la question « Que se passe-t-il si les canaux veulent interrompre tous les deux en même temps ? » Les interruptions font l'objet d'une priorité interne dans le SIO. Voici la liste des bits possibles du vecteur (D3, D2, D1) énumérés dans l'ordre des priorités, de la plus haute à la plus basse.

D3	D2	D1	TYPE D'INTERRUPTION	CANAL UTILISE
1	1	1	condition de réception spéciale	canal A
1	1	0	caractère reçu	canal A
1	0	0	tampon de transmission vide	canal A
1	0	1	transition externe/état	canal A
0	1	1	condition de réception spéciale	canal B
0	1	0	caractère reçu	canal B
0	0	0	tampon de transmission vide	canal B
0	0	1	transition externe/état	canal B

Supposons par exemple, que nous ayons programmé le vecteur d'interruption 10010000. Supposons, de plus, que les interruptions soient validées et que survienne une interruption. Le canal A a reçu un caractère. Le vecteur d'interruption résultant, qui serait placé dans le bus de données pendant une séquence de réception d'interruption, serait 10011100. Remarquez que les bits D3, D2 et D1 ont été modifiés par rapport au vecteur d'origine pour montrer l'état de l'interruption (c'est-à-dire pour montrer pourquoi l'interruption s'est produite).

Regardons maintenant de plus près la signification des conditions d'interruption. Nous allons commencer par la condition de réception spéciale. Les conditions comportent les erreurs de parité, les surcharges de récepteur, les erreurs de cadrage, ou fin de cadrage. Dans l'exposé précédent, nous avons appris que nous pouvons tester les résultats de ces types d'erreurs en lisant RR1.

Le vecteur d'interruption suivant est un caractère reçu disponible. Ce mode cause une interruption si le tampon du récepteur a au moins un caractère disponible pour lecture par le CPU. Il faut remarquer que lorsqu'un caractère est reçu, il est possible d'avoir les conditions de réception spéciale actives, indiquant ainsi qu'une erreur s'est produite dans la réception du caractère. On a maintenant deux vecteurs d'interruption possibles. Dans cet exemple, le vecteur d'interruption des conditions de réception spéciale aurait la priorité interne sur le vecteur d'interruption de caractère reçu.

Le tampon de transmission vide vient ensuite. Ce vecteur d'interruption a lieu quand le tampon de transmission est prêt à transmettre un autre caractère.

Enfin, le vecteur d'interruption de transition d'état externe a lieu chaque fois qu'il y a un changement dans les lignes d'état de CTS, DCD et SYNC.

11.10 - Initialisation du SIO pour les interruptions

La séquence d'initialisation du SIO lorsque l'on utilise les interruptions est très semblable à l'initialisation quand on n'utilise pas d'interruption. Il y a deux registres qui doivent être employés et qui ont peu d'importance pour la scrutation. Ce sont WR2 (vecteur d'interruption) et WR1 (mode interruption). Dans l'initialisation pour scrutation, il suffit d'écrire un octet dans WR1 pour inhiber les interruptions.

Pour montrer comment le SIO peut être initialisé pour des interruptions, examinons un exemple. Nous allons d'abord configurer le SIO avec les mêmes caractéristiques de fonctionnement qui étaient établies en mode à scrutation. La seule différence est que le système d'interruption sera validé. Nous allons utiliser le canal B comme canal de communication. La Figure 11.12 donne le programme Z80 d'initialisation.

11.11 - Après l'initialisation

Une fois le SIO initialisé, il est prêt à fonctionner avec des interruptions. Cependant, des routines spéciales doivent être écrites pour traiter chaque type d'interruption que le SIO peut produire par programme. Pour activer l'interrupteur récepteur plein, il suffit que le caractère soit lu dans le registre de données du canal — ceci va vider le tampon et ôter la source de l'interruption. L'instruction RETI doit ensuite être exécutée pour remettre à zéro la demande d'interruption hardware.

```
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   CE PROGRAMME INITIALISE SIO AVEC INTERRUPTION
;   VECTEUR D'INTERRUPTION : 98H
;   CANAL B : CANAL ACTIF
;   ADRESSAGE SIO : 8C-8F
;
;   PARAMETRES PRINCIPAUX : HORLOGE X64, 2 BITS D'ARRET,
;   8 BITS RX, 8 BITS TX, PAS DE PARITE.
;   INT. SUR RX, TX, ETAT AFFECTE VECTEUR
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
008D              BDATA  EQU  8DH        ;PORT DE DONNEES DU CANAL B
008F              BCON   EQU  8FH        ;PORT DE COMMANDE DU CANAL B
;
;
1800   3E18              LD A,18H
1802   D38F              OUT (BCON),A    ;WR0, RAZ CANAL
1804   3E02              LD A,02H
1806   D38F              OUT (BCON),A    ;WR0, POINTER WR2
1808   3E98              LD A,98H
180A   D38F              OUT (BCON),A    ;WR2, INT. VECT = 98H
180C   3E04              LD A,04H
180E   D38F              OUT (BCON),A    ;WR0, POINTER WR4
1810   3ECC              LD A,0CCH
1812   D38F              OUT (BCON),A    ;WR4, HORL. X64, 2 BITS D'ARRET, NON
1814   3E03              LD A,03H
1816   D38F              OUT (BCON),A    ;WR0, POINTER WR3
1818   3EC1              LD A,0C1H
181A   D38F              OUT (BCON),A    ;WR3, 8 BITS RX, VALIDATION RX
181C   3E05              LD A,05H
181E   D38F              OUT (BCON),A    ;WR0, POINTER WR5
1820   3E68              LD A,68H
1822   D38F              OUT (BCON),A    ;WR5, 8 BITS TX, VALIDATION TX
1824   3E01              LD A,01H
1826   D38F              OUT (BCON),A    ;WR0, POINTER WR1
1828   3E16              LD A,16H
;
182A   D38F              OUT (BCON),A    ;WR1, INT. RX, ETAT TX
182C   C9                RET
;
;   FIN DE LA ROUTINE D'INITIALISATION
;
                         END
```

Figure 11.12 : *Programme Z80 pour initialiser le Z80-SIO dans une application d'interruption.*

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;   CETTE ROUTINE DE SERVICE SERT
;   A RECEVOIR UN CARACTERE
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
008D                BDATA  EQU   8DH       ;REG. DE DONNEES DU CANAL B
008F                BCON   EQU   8FH       ;REG. DE COMMANDE DU CANAL B
5900                RXCH   EQU   5900H     ;EMPLACEMENT POUR CARACTERE REÇU
5901                TXCH   EQU   RXCH+1    ;EMPLACEMENT CARACTERE TX
;
1800    F5          RXINT  PUSH AF         ;SAUVEGARDER REGISTRES
1801    DB8D               IN A,(BDATA)    ;LIRE CARACTERE
1803    320059             LD (RXCH),A     ;SAUVEGARDER CARACTERE
1806    F1                 POP AF          ;RESTAURER REGISTRES
1807    FB                 EI              ;VALIDER INTERRUPTIONS
1808    ED4D               RETI            ;RETOUR D'INTERRUPTIONT
;
;
;  VOICI UNE PETITE ROUTINE QUI MANIPULE
;  INT. TX VIDE QUAND IL N'Y A PLUS
;  DE CARACTERE A ENVOYER
;
180A    F5          TXINT  PUSH  AF        ;SAUVEGARDER REGISTRE
180B    3E28               LD A,28H
180D    D38F               OUT (BCON),A    ;WRO, RESET INTERRUPTION TX
180F    F1                 POP   AF
1810    FB                 EI
1811    ED4D               RETI
;
; VOICI TRAITEMENT INT. TX VIDE
; QUAND IL N'Y A PLUS DE CARACTERE A ENVOYER
;
1813    F5          TXINT1 PUSH  AF
1814    3A0159             LD A,(TXCH)      ;LIRE CARACTERE A ENVOYER
1817    D38D               OUT (BDATA),A    ;ENVOYER CARACTERE AU SIO
1819    F1                 POP   AF

181A    FB                 EI
181B    ED4D               RETI
;
;
                           END
```

***Figure 11.13 :** Exemple de deux routines de service d'interruption Z80 avec Z80-SIO pour transmettre et recevoir un caractère.*

```
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  ROUTINE DE SERVICE D'INTERRUPTION
;  EN CONDITION SPECIALE DE RECEPTION
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
008D                BDATA  EQU  8DH        ;PORT DE DONNEES DU CANAL B
008F                BCON   EQU  8FH        ;PORT DE COMMANDE DU CANAL B
;
1B00   DB8D                IN A,(BDATA)    ;LIRE CARACTERE D'ENTREE
;
;  CARACTERE D'ENTREE DOIT ETRE LU POUR RAZ DU TAMPON D'ENTREE.
;  LES DONNEES NE DOIVENT PAS ETRE EMPLOYEES A CAUSE ERREUR.
;
1B02   3E01                LD A,01H
1B04   D38F                OUT (BCON),A    ;WRO, POINTER RR1
;
;  NOUS LISONS RR1 POUR DETERMINER QUEL TYPE D'ERREUR A
;  ENTRAINE L'INTERRUPTION ;LECTURE TYPE ERREUR
;
1B06   DB8F                IN A,(BCON)
;
;  BITS D6, D5, D4 INDIQUENT TYPE D'ERREUR
;
;  ICI REG. D'ERREUR PEUT ETRE SAUVEGARDE
;  POUR TRAITEMENT ULTERIEUR.
;  NON EFFECTUE ICI.
;
1B08   3E30                LD A,30H
1B0A   D38F                OUT (BCON),A    ;WRO, COMMANDE RESET ERREUR
;
;  OUT PRECEDENT CAUSE RAZ ERREURS.
;  ON PEUT DONC LES EXAMINER
;  POUR LE CARACTERE REÇU SUIVANT.
;
;  ON DOIT MAINTENANT RE-VALIDER LES INTERRUPTIONS
;
1B0C   FB                  EI
;
;
1B0D   ED4D                RETI
;
;  L'INSTRUCTION RET EST NECESSAIRE POUR INITIALISER
;  LES INTERRUPTIONS POUR LE CIRCUIT DU SIO.
;
                           END
```

Figure 11.14 : *Programme Z80 montrant une routine d'interruption pour servir des conditions de réception spéciales avec Z80-SIO.*

Quand une interruption est causée par la condition transmetteur vide, une commande « remettre à zéro interruption Tx » est produite et le caractère Tx suivant est mis dans le SIO. La Figure 11.13 montre un exemple de deux petites routines d'interruption qui vont traiter les conditions d'interruption de réception et de transmission. Ces routines feront généralement partie d'une routine de service plus grande. Elles mettent en évidence plusieurs points importants concernant la gestion d'interruptions avec SIO.

Enfin, la Figure 11.14 montre une routine d'interruption qui peut être employée pour gérer les conditions de réception spéciale.

RESUME

Dans ce chapitre, nous avons examiné le fonctionnement et la programmation du SIO. Nous avons étudié un schéma de ce composant et avons appris à le programmer. Les renseignements donnés dans ce chapitre et dans l'Annexe devraient vous documenter suffisamment pour vous permettre d'employer le SIO de la façon convenant à vos applications.

ESSAIS DU Z80 PAR STIMULATION STATIQUE

CHAPITRE 12

INTRODUCTION

Dans ce dernier chapitre, nous allons parler d'une technique de mise au point et de dépannage de circuit appelée *Static Stimulus Testing (SST)* ou *essais par stimulation statique*. Créée à l'origine pour l'industrie, cette technique permet de vérifier, d'une façon simple et peu coûteuse, les circuits de votre système sans avoir à utiliser de logiciel. On peut employer cette technique pour vérifier toutes les interfaces externes entre système à Z80 et dispositifs LSI que nous avons examinés dans ce livre.

Le SST a été créé pour les réparations et la mise au point des systèmes à microprocesseurs ; ceux-ci peuvent être classés en deux catégories :

- Systèmes ayant subi une panne destructive et qui ne peuvent exécuter aucun logiciel (c'est-à-dire systèmes ayant fonctionné).

- Systèmes au stade prototype et n'ayant jamais fonctionné. Il s'agit de systèmes qui n'ont pas encore été mis au point ou bien de systèmes pour lesquels le logiciel n'est pas complet et qui, par conséquent, ne peuvent pas être employés pour faire de diagnostics. Le concepteur des circuits ou le dépanneur peuvent souhaiter vérifier le bon fonctionnement de l'ensemble ou d'une partie des circuits de ces systèmes sans utiliser le logiciel.

Dans ces deux catégories, les techniques de dépannage hardware classiques, telles que l'analyse d'état logique et l'analyse de signature, offrent peu d'assistance : avec ces deux techniques, il faut que le système puisse exécuter du logiciel si l'on veut qu'elles soient utiles. Bien que l'on mette au point de nouvelles techniques d'analyse de signature qui éliminent cette sujétion, il y a encore un grand fossé à combler au point de vue dépannage des systèmes à microprocesseurs.

Une question importante est la suivante : « Par où et de quelle façon commence-t-on à mettre au point un système qui ne fonctionne pas du tout ? » Autrement dit, « Comment met-on au point un système qui ne peut exécuter aucun logiciel ? »

L'essai par stimulation statique, bien que dans ses débuts, comble en fait ce fossé. Avec le SST on peut mettre au point un système qui ne fonctionne pas, d'une façon directe et ordonnée, jusqu'à ce que l'on puisse procéder au diagnostic logiciel. Ceci peut être effectué d'une manière efficace et directe, en employant des appareils simples et peu coûteux.

Le SST a pour principal caractère d'être totalement indépendant du logiciel du système. Donc, un technicien ou un ingénieur ayant peu d'expérience en logiciel, peuvent utiliser facilement le SST. En outre, une personne experte en logiciel, mais ayant peu d'expérience en technique des circuits, pourra apprendre facilement le principe du SST.

Nous allons donner des exemples d'utilisation du SST pour vérifier les circuits d'un système complet. Nous supposerons que le logiciel n'existe pas. En d'autres termes, nous supposerons que nous avons un système prototype qui vient d'être construit et que nous voulons vérifier le fonctionnement de ses circuits, ou bien que le système a mal fonctionné et qu'il n'y a pas de listing de logiciel (ou peut-être que l'on a peu de connaissance du logiciel).

En commençant à employer la technique SST pour vérifier chaque partie des circuits, vous acquerrez un certain « flair » sur ce qui se passe. En fait, l'emploi de ce moyen vous permettra de mieux comprendre comment le Z80 communique électriquement.

12.1 - Aperçu des essais par stimulation statique

L'idée essentielle de ce genre d'essai repose sur le fait que les communications électriques qui ont lieu dans un système à microprocesseur sont de nature essentiellement statique. Autrement dit, il y a deux niveaux de tension, l'un représentant des 1, l'autre des 0. Dans un système à microprocesseur, on trouve alternativement ces deux états. Les événements électriques ont généralement lieu en successions rapides, mais ce n'est pas obligatoire. Il y a, en fait, une limite supérieure à la vitesse de fonctionnement d'un système, mais il n'y a pas, en principe, de limite inférieure.

Dans un système à microprocesseur, non seulement les communications entre microprocesseur, mémoires et dispositifs d'entrée-sortie sont électriquement statiques, mais les lignes des signaux effectuent aussi une fonction électrique distincte à tout instant. Par exemple, pendant une communication avec des mémoires, les lignes d'adresses pointent un emplacement mémoire particulier. Ceci a lieu quels que soient les états logiques des autres signaux. Quand on emploie le SST, chaque ligne de signal peut être considérée comme indépendante.

Chaque signal a *toujours* un point d'origine électrique et un point de destination. En employant le SST comme source, on peut, au point

d'origine, forcer une ligne de signal distincte à l'état logique désiré. On peut ensuite tracer statiquement la réponse électrique de la ligne. (*Note :* Le SST ne tient pas compte de l'élément temps.)

En utilisant les techniques de dépannage digital standard et le SST, on peut mettre au point les circuits d'un système complet. Quelle que soit la complexité des circuits, une fois l'état dynamique transformé en état statique, les zones présentant des difficultés peuvent être trouvées beaucoup plus facilement.

Un autre avantage du SST est qu'il fonctionne avec des systèmes contenant des composants LSI spéciaux, tels que PIO ou SIO. Ceci est dû à son fonctionnement basé sur la nature statique des communications du microprocesseur.

Il se peut aussi que vous contestiez ce fait en faisant remarquer que les RAM dynamiques ne sont pas des éléments statiques. Bien que ce soit théoriquement vrai, il faut l'interpréter. La seule partie qui est dynamique est la cellule de stockage de la mémoire. Toutes les entrées d'adresses, $\overline{RAS}$, $\overline{CAS}$ et MUX, peuvent être considérées comme fonctionnant d'une façon statique. Donc, il est exact que la cellule mémoire elle-même ne fonctionne pas en mode statique, mais on peut employer une approche statique pour analyser tous les signaux périphériques, de même que les circuits du système mémoire.

Pour expliquer les points essentiels du SST, étudions un exemple. Dans cet exemple, nous allons vérifier les entrées d'adresses des ROM du système. (*Note :* Il n'est pas possible de donner ici la procédure d'utilisation détaillée du SST. Le but principal est de vous montrer comment le principe du SST peut être appliqué à un problème courant.)

La Figure 12.1 montre le schéma de la ROM dans un système à microprocesseur. Cette figure indique aussi les points de départ et d'arrivée des lignes d'adresses de la ROM. Nous allons employer cet exemple pour vérifier au moyen du SST si ces lignes d'adresses fonctionnent correctement, au point de vue matériel. Pour atteindre cet objectif, nous allons suivre les cinq étapes suivantes :

1. Nous ôtons pour commencer le microprocesseur du système et installons un câble connecté au panneau de commande du SST comme le montre la Figure 12.2.

2. Supposons que nous ayons un contact sur chaque ligne d'adresses A0 à A15. Chacun d'eux peut forcer un 1 ou un 0 logiques dans une ligne d'adresse. Ces 16 contacts peuvent être utilisés par l'opérateur pour établir les 64 K combinaisons logiques différentes possibles.

3. La combinaison établie par ces contacts d'adresses peut être maintenue indéfiniment (statique). Nous pouvons maintenant vérifier les entrées des tampons d'adresses d'un système à Z80 en utilisant des techniques statiques de mesure en courant continu. Nous pouvons déterminer si tous les points de ces lignes montrent la même condition logique (appropriée) que celle établie par les contacts. De plus, les sorties des tampons d'adresses doivent être identiques aux lignes d'entrées A0-A15 qui ont été établies par les contacts d'adresses du SST.

4. Nous examinons ensuite les niveaux logiques des lignes d'adresses aux entrées d'adresses des ROM. Pour cela, on peut utiliser un oscilloscope, une sonde logique, un voltmètre digital ou tout autre appareil de mesure de courant continu.

5. Toute logique de décodage d'adresse peut aussi être vérifiée, à ce moment-là, d'une façon statique.

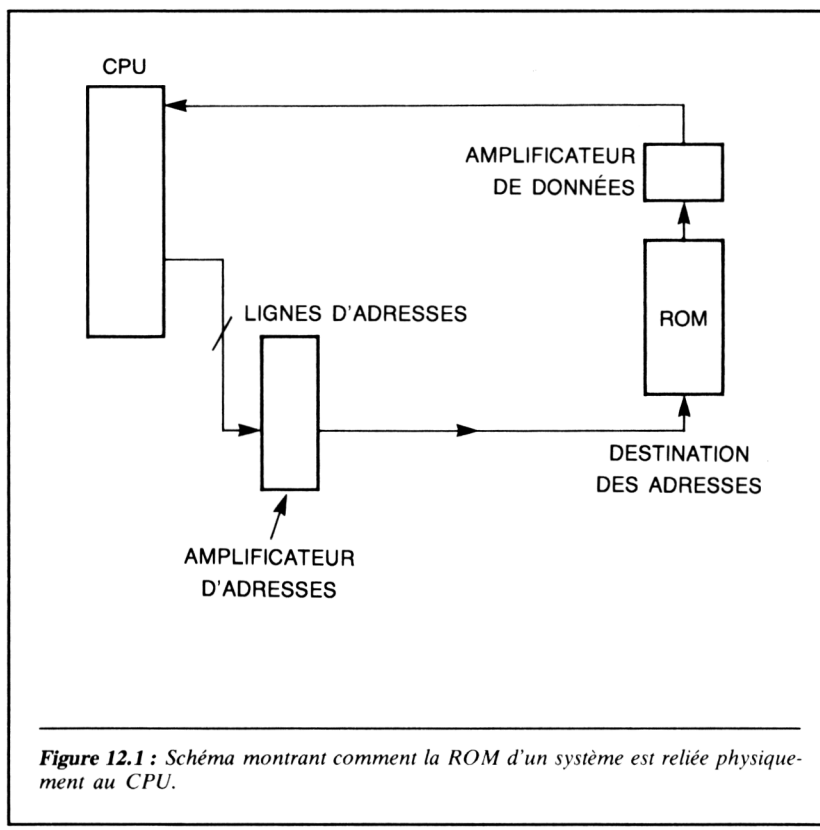


Figure 12.1 : *Schéma montrant comment la ROM d'un système est reliée physiquement au CPU.*

Dans cet exemple général, nous n'avons examiné que les lignes d'adresses. Le SST permet de les examiner séparément parce qu'il détermine une indépendance dans les signaux.

Remarquez aussi que nous n'avons pas mentionné le temps absolu : ce qui compte avec le SST, c'est la séquence des états électriques ; on peut prendre tout le temps nécessaire pour suivre un signal depuis son origine jusqu'à sa destination.

12.2 - Circuits du testeur statique à stimulation

Le SST est un dispositif à circuit très simple. Il convient aussi bien à des buts éducatifs qu'à des buts industriels. L'idée est la suivante : l'opérateur a

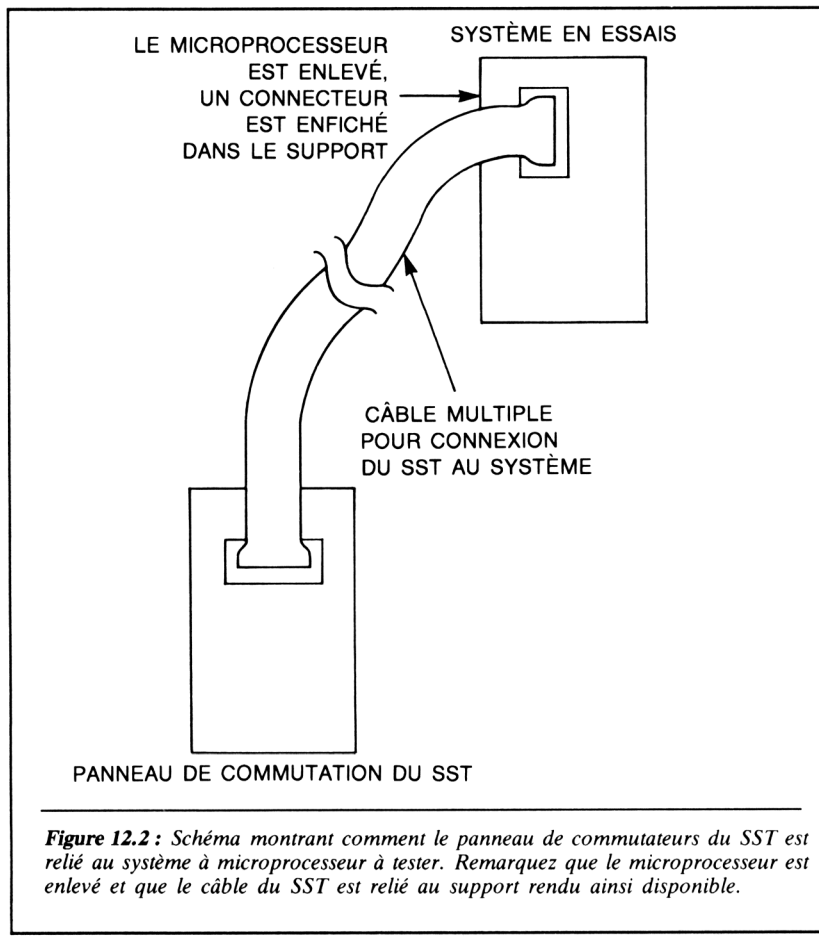


Figure 12.2 : *Schéma montrant comment le panneau de commutateurs du SST est relié au système à microprocesseur à tester. Remarquez que le microprocesseur est enlevé et que le câble du SST est relié au support rendu ainsi disponible.*

le contrôle statique des niveaux logiques de toutes les lignes de signaux qui devraient être normalement contrôlées par le microprocesseur. Pour le Z80, ces lignes sont les suivantes :

A0-A15	$\overline{\text{IORQ}}$
D0-D7	$\overline{\text{RFSH}}$
$\overline{\text{RD}}$	$\overline{\text{HALT}}$
$\overline{\text{WR}}$	$\overline{\text{M1}}$
$\overline{\text{MREQ}}$	$\overline{\text{BUSAK}}$

Les autres broches du Z80 sont des entrées dont nous n'avons pas à nous occuper pour le moment.

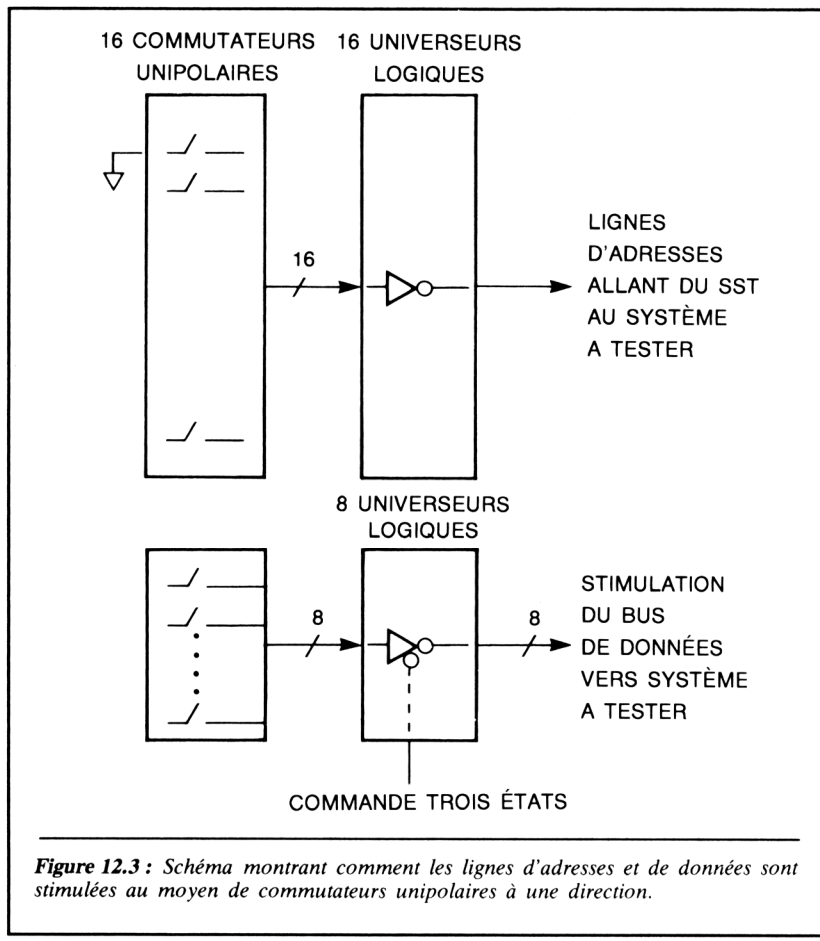


Figure 12.3 : *Schéma montrant comment les lignes d'adresses et de données sont stimulées au moyen de commutateurs unipolaires à une direction.*

En remarque finale, il est important de se rappeler que l'emploi du SST va couper toutes les boucles de réaction du système.

Examinons maintenant les circuits du SST.

12.3 - Lignes de sortie d'adresses et de données du SST

La Figure 12.3 montre le schéma permettant au SST de réaliser la stimulation d'adresses et de données. Dans cette figure, la valeur logique

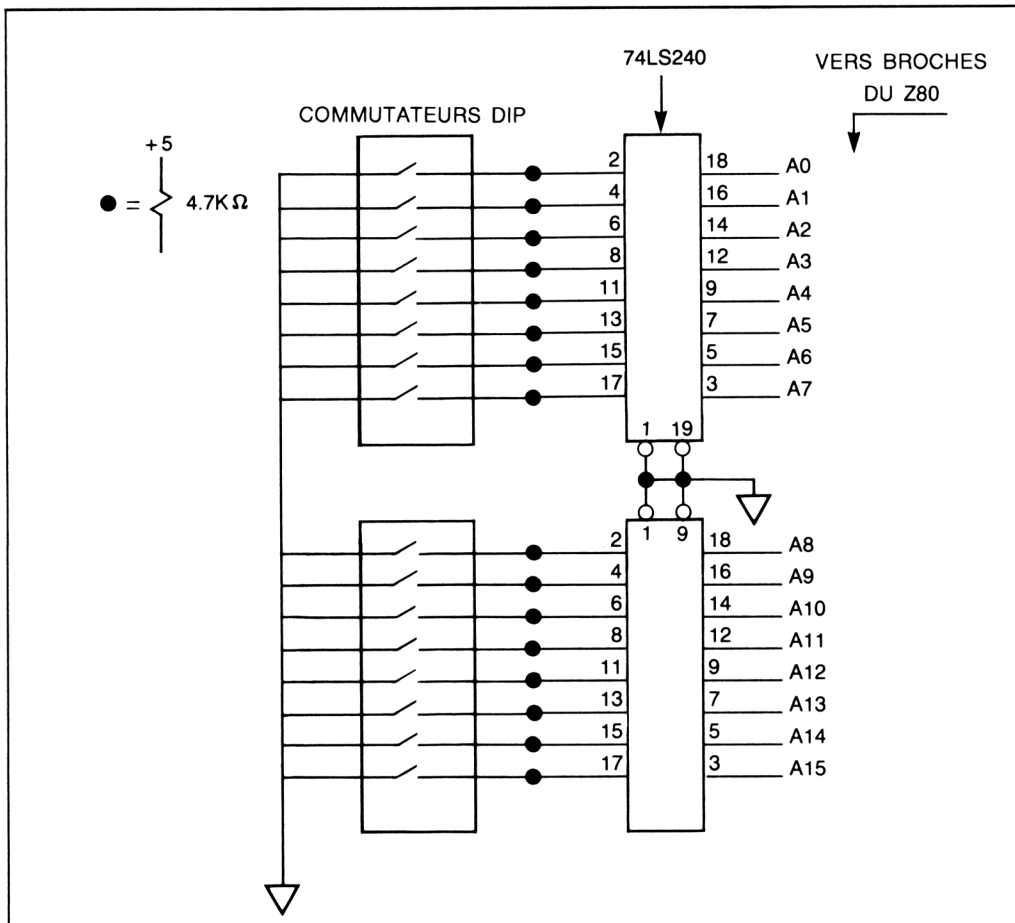


Figure 12.4 : *Schéma montrant comment les sorties d'adresses sont mises au 0 ou au 1 logique.*

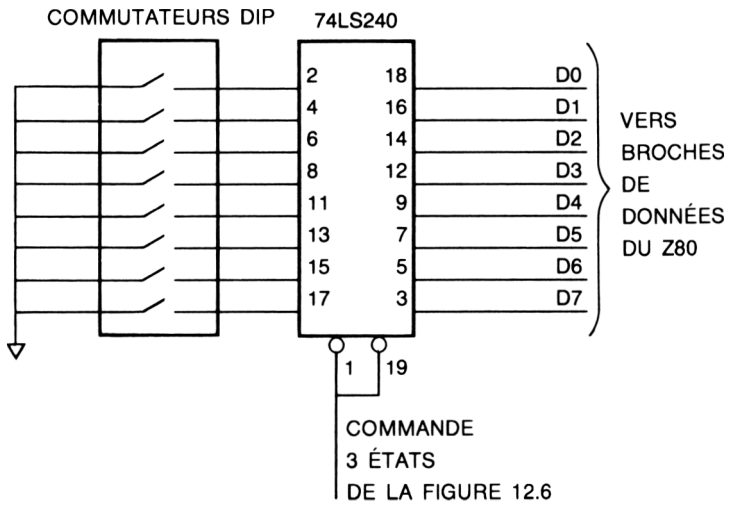


Figure 12.5 : *Schéma montrant comment les lignes de données sont mises au 1 logique ou au 0 logique.*

des sorties d'adresses est déterminée par la position physique des contacts DIP. La sortie de chacun de ceux-ci est inversée et amplifiée. Les amplificateurs utilisés pour les stimulations de données ont une commande trois états (nous en expliquerons la raison plus loin). Il suffit de savoir pour le moment que les amplificateurs de données peuvent être mis en trois états.

Les sorties des amplificateurs référencées D0-D7 servent à réaliser les lignes de données du système. Quand le SST sert à dépanner le système à Z80, les contacts DIP sont employés pour stimuler les sorties de données en les utilisant de la même façon que le Z80. Les Figures 12.4 et 12.5 montrent le schéma effectif permettant de réaliser les parties stimulation d'adresses et de données du SST.

12.4 - $\overline{M1}$, $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, $\overline{WR}$, $\overline{RFSH}$, $\overline{HALT}$, $\overline{BUSAK}$

Nous allons maintenant apprendre à produire, au moyen du SST, les signaux de commande suivante :

$\overline{M1}$	$\overline{WR}$
$\overline{MREQ}$	$\overline{RFSH}$
$\overline{IORQ}$	$\overline{HALT}$
$\overline{RD}$	$\overline{BUSAK}$

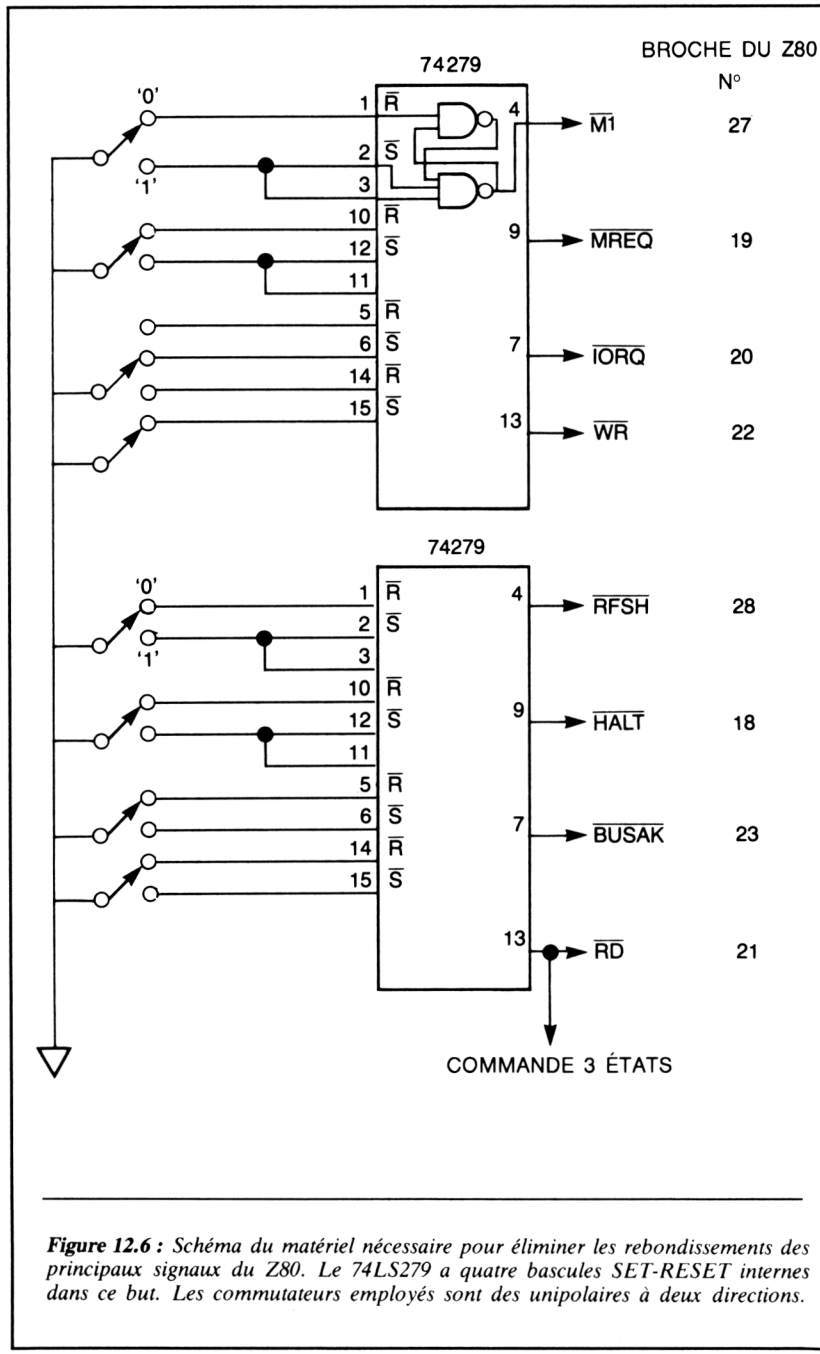


Figure 12.6 : *Schéma du matériel nécessaire pour éliminer les rebondissements des principaux signaux du Z80. Le 74LS279 a quatre bascules SET-RESET internes dans ce but. Les commutateurs employés sont des unipolaires à deux directions.*

Nous examinerons tous ces signaux dans la même section parce que les circuits nécessaires pour chacun d'eux sont identiques.

Tous les signaux énumérés ci-dessus sont des bits de commandes produits par le Z80 et émis vers le système. La plupart sont valides dès le début d'un cycle de mémoire et le restent pendant un cycle complet. Pour produire ces signaux, on utilise, pour les stimulis, des commutateurs unipolaires à deux directions. Les sorties de ceux-ci sont appliquées à des circuits anti-rebonds dont les sorties sont finalement reliées au système à tester. La Figure 12.6 montre le schéma correspondant.

Ce schéma utilise le circuit intégré 74279. Celui-ci contient quatre bascules S-R (*Set-Reset*), qui servent à supprimer les rebondissements des contacts.

12.5 - Affichage par LED pour le bus de données

Nous avons vu tous les circuits du SST, excepté ceux permettant d'analyser le bus de données du système. Cet examen peut être effectué au

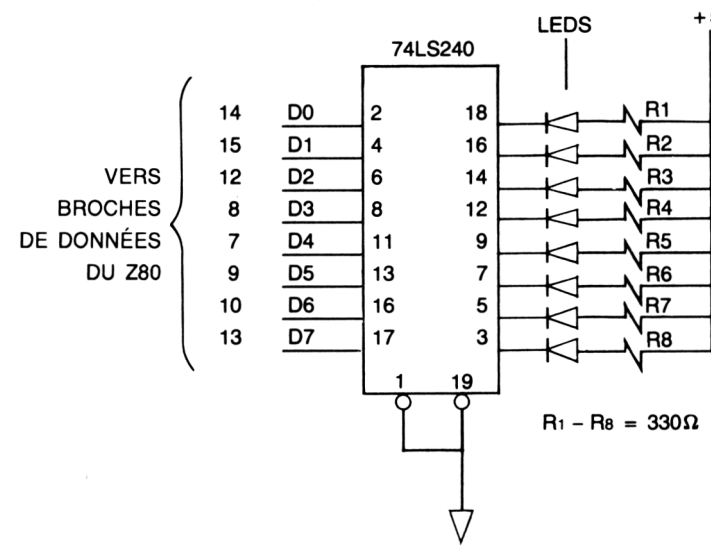


Figure 12.7 : *Schéma montrant comment les LED du SST indiquent les niveaux logiques du bus de données du système.*

moyen de huit LED affichant les conditions logiques des lignes du bus de données du système. La Figure 12.7 montre cet affichage. Le bus de données du microprocesseur (ou du SST) est connecté aux entrées des inverseurs 74LS240. Les sorties de ceux-ci alimentent les LED. Un 0 logique aux entrées du 74LS240 éteint les LED, un 1 logique les allume.

Les LED représentent les conditions logiques des lignes de signaux aux broches D0-D7 du microprocesseur. Ceci permet de voir effectivement la valeur logique en entrée ou en sortie du microprocesseur. Si deux lignes de données sont mélangées ou bien si les données n'atteignent pas les broches d'entrées du microprocesseur, cela se verra tout de suite sur les LED.

Pendant une opération d'écriture, on peut facilement voir quelles sont les données que le Z80, ou le SST, émet vers les circuits du système. Remarquez que cet affichage montre directement les données aux broches du CPU. Ceci est utile lorsque l'on vérifie les lignes du bus de données du système.

RESUME

Dans ce chapitre, nous avons examiné les circuits permettant d'employer la technique d'essais par stimulation statique (SST) avec le Z80. Cette technique constitue un outil puissant pour mettre au point un système à microprocesseur. On peut l'employer pour mettre au point des prototypes, aussi bien que pour dépanner des systèmes produits en série. Le SST peut être employé par n'importe qui. Il n'a pas besoin de spécialiste hautement qualifié pour donner d'excellents résultats.

Pour terminer, nous dirons que si vous n'avez pas de testeur par stimulation statique et que vous désiriez en acquérir un, vous avez deux options. Vous pouvez soit en construire un à partir des schémas de ce chapitre, soit l'acheter à un constructeur*.

---

* Il existe deux versions du Z80 SST chez Creative Microprocessor Systems, Inc., P.O. Box 1538, LOS GATOS, CA. 95030.

DESCRIPTION DES REGISTRES INTERNES DU Z80-SIO

ANNEXE

REGISTRES D'ECRITURE

Le SIO contient huit registres (WR0-WR7) par canal ; ces registres sont programmés séparément par le programme du système pour mettre chaque canal dans sa configuration spécifique. A l'exception de WR0, la programmation des registres d'écriture nécessite deux octets. Les trois bits du premier octet (D_0-D_2) pointent le registre choisi ; le second octet est le mot de commande effectif écrit dans le registre pour configurer le SIO.

Remarquez que le programme, après pointage du registre choisi, est complètement libre, soit de lire pour tester le registre de lecture, soit d'écrire pour initialiser le registre d'écriture. En élaborant le logiciel pour initialiser le SIO d'une façon modulaire et structurée, le programmeur peut utiliser des blocs d'instructions d'entrée-sortie puissants.

WR0 est un cas spécial, car on peut accéder à toutes les commandes de base (CMD_0-CMD_2) avec un seul octet. Reset (interne ou externe) initialise les bits du pointeur (D_0-D_2) pour qu'ils pointent vers WR0.

Les commandes fondamentales (CMD_0-CMD_2) et les commandes CRC (CRC_0-CRC_1) sont contenues dans le premier octet de n'importe quel accès de registre d'écriture. Ceci permet un maximum de souplesse et de commande du système. Chaque canal contient les registres de commande suivants. Ces registres sont adressés comme des commandes (pas comme des données).

REGISTRE D'ECRITURE 0

WR0 est le registre de commande ; il sert aussi aux codes d'initialisation CRC et pour pointer les autres registres.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CRC Code d'initialisation 1	CRC Code d'initialisation 0	CMD 2	CMD 1	CMD 0	PTR 2	PTR 1	PTR 0

Bits de pointeur (D_0-D_2). Les bits D_0-D_2 sont les bits de pointeur qui déterminent dans quel autre registre l'octet suivant doit être écrit ou dans quel registre de lecture l'octet suivant doit être lu. Le premier octet écrit dans chaque canal après une initialisation (soit par une commande d'initialisation soit par l'entrée d'initialisation extérieure) va dans WR0. Après une lecture ou une écriture dans n'importe quel registre (excepté WR0),le pointeur pointe WR0.

Bits de commande (D_3-D_5). Trois bits, D_3-D_5, sont codés pour émettre les sept commandes de base du SIO.

COMMANDE	CMD_2	CMD_1	CMD_0	
0	0	0	0	Commande nulle, sans effet
1	0	0	1	Envoyer Abort (mode SDLC)
2	0	1	0	Initialisation interruptions externe/état
3	0	1	1	Initialisation de canal
4	1	0	0	Valider interruption sur caractère Rx suivant
5	1	0	1	Initialiser interruption de transmetteur
6	1	1	0	Initialiser erreur *(latch)*
7	1	1	1	Retour d'interruption (canal A)

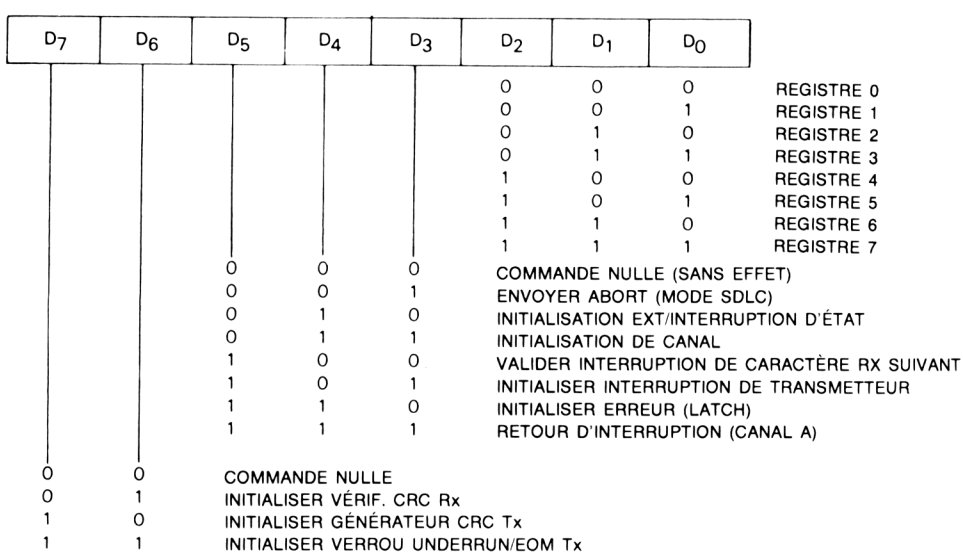
Commande 0 (Nulle). La commande Null n'a pas d'effet. Quand on l'utilise, le SIO ne fait rien pendant que les pointeurs sont établis pour l'octet suivant.

Commande 1 (Envoyer arrêt). Cette commande n'est utilisée qu'avec le mode SDLC pour produire une séquence de huit à treize 1.

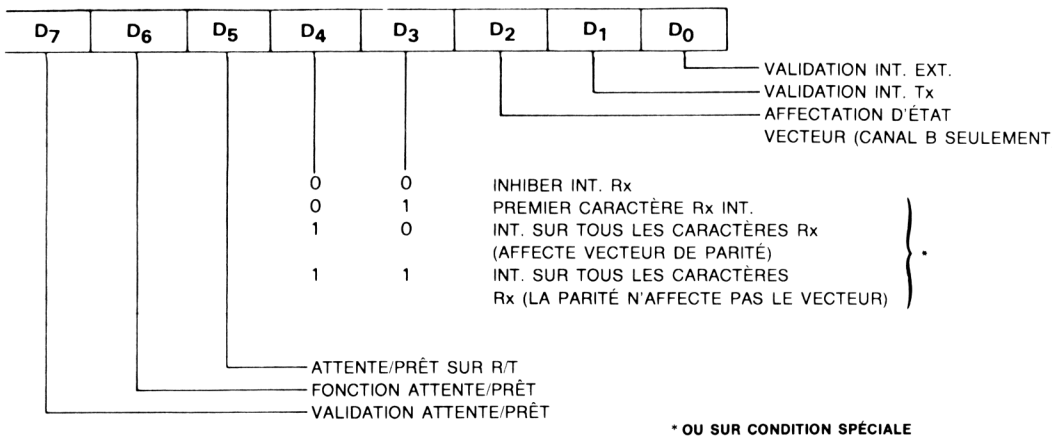
Commande 2 (Initialisation Interruptions Externe/Etat). Après une interruption Externe/Etat (changement dans une ligne de modem ou condition d'arrêt, par exemple) les bits d'état de RR0 sont verrouillés. Cette commande les valide à nouveau et permet de nouvelles interruptions. En verrouillant les bits d'état, on saisit de courtes impulsions jusqu'à ce que le CPU ait le temps de lire le changement.

FONCTIONS DES BITS DU REGISTRE D'ÉCRITURE

REGISTRE D'ECRITURE 0

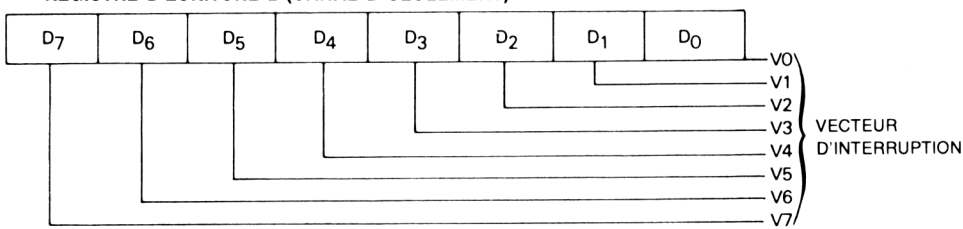


REGISTRE D'ECRITURE 1

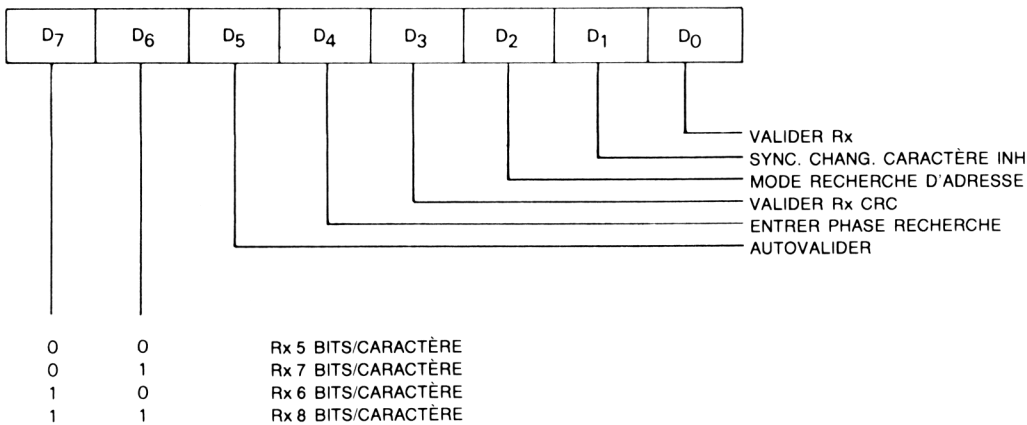


* OU SUR CONDITION SPÉCIALE

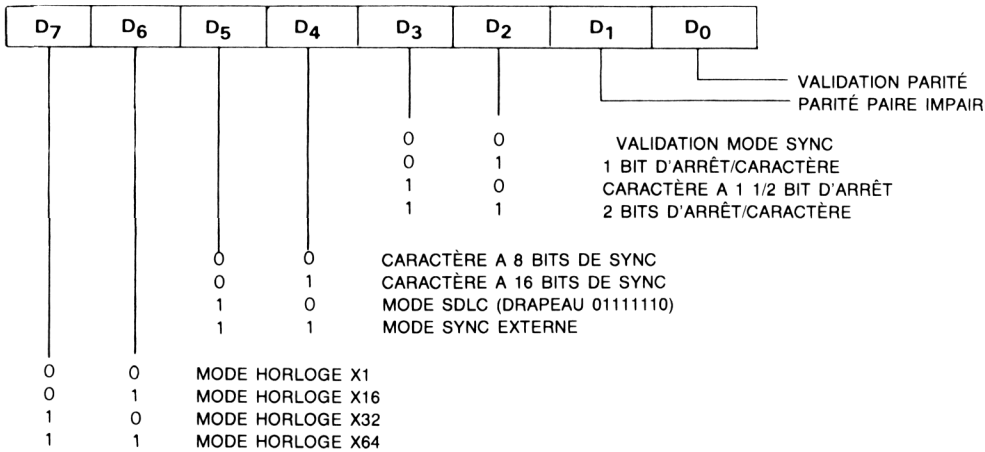
REGISTRE D'ECRITURE 2 (CANAL B SEULEMENT)



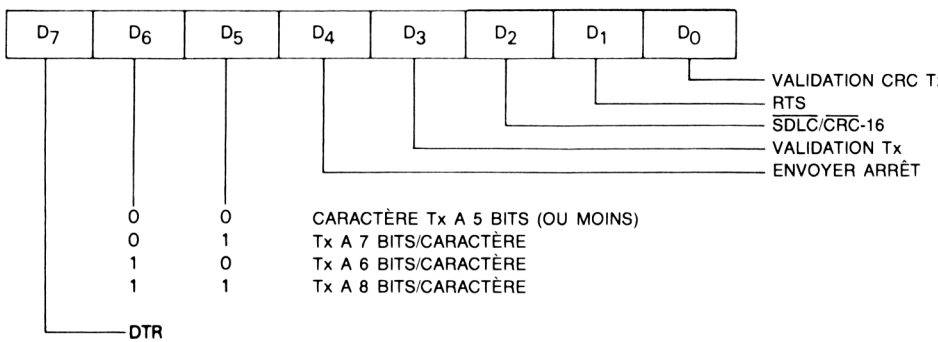
REGISTRE D'ECRITURE 3



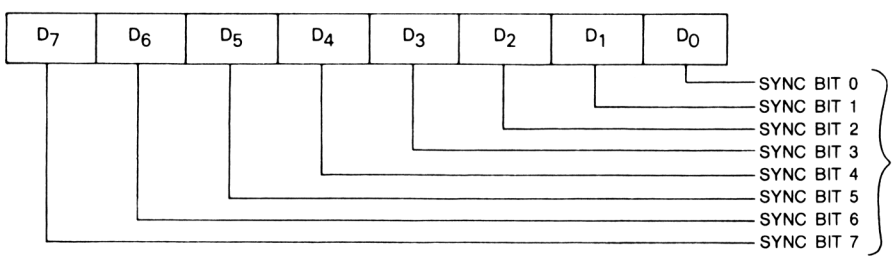
REGISTRE D'ECRITURE 4



REGISTRE D'ECRITURE 5

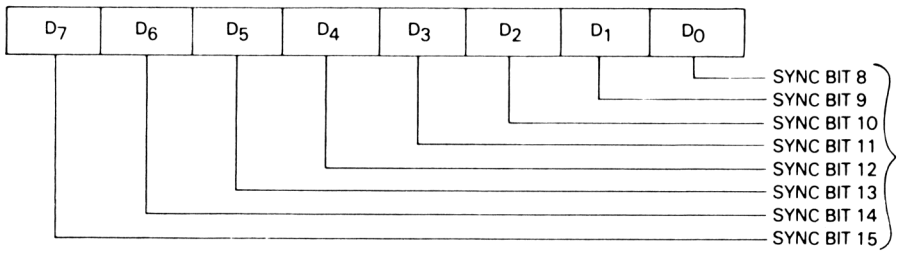


REGISTRE D'ECRITURE 6



*** CHAMP D'ADRESSES SDLC AUSSI**

REGISTRE D'ECRITURE 7



*** POUR SDLC, DOIT ÊTRE PROGRAMMÉ A 01111110 POUR RECONNAISSANCE DE DRAPEAU**

Commande 3 (Initialisation de canal). Cette commande effectue les mêmes fonctions qu'une initialisation externe, mais seulement sur un seul canal. L'initialisation du canal A initialise aussi la logique de priorité d'interruption. Tous les registres de commande du canal doivent être écrits à nouveau après une commande d'initialisation de canal.

Après une initialisation de canal, il faut quatre cycles d'horloge de système supplémentaires pour le temps d'initialisation du SIO avant toute nouvelle écriture de commande dans ce canal. Ceci peut être, en principe, le temps employé par le CPU pour rechercher le code opération suivant.

Commande 4 (Validation d'interruption sur caractère suivant). Si l'interruption sur le mode premier caractère reçu est sélectionnée, cette commande réactive ce mode après réception de chaque message complet pour préparer le SIO au prochain message.

Commande 5 (Initialisation interruption du transmetteur). Le transmetteur interrompt quand le tampon de transmission devient vide, si l'on choisit le mode de validation d'interruption de transmission. Au cas où il n'y a plus de caractère à envoyer (à la fin d'un message par exemple), l'émission de cette commande empêche d'autres interruptions de transmetteur jusqu'à ce que le caractère suivant ait été chargé dans le tampon de transmission, ou jusqu'à ce que CRC ait été complètement envoyé.

Codes 0 et 1 d'initialisation de CRC (D_6 et D_7). Ces bits sélectionnent ensemble l'une des trois commandes d'initialisation suivantes :

Initialisation CRC Code 1	Initialisation CRC Code 0	
0	0	Code nul (sans effet)
0	1	Initial. Vérif. CRC réception
1	0	Initial. générateur CRC transmission
1	1	Initial. sous charge Tx/latch de fin de message

La commande de générateur CRC transmetteur initialisation initialise en principe le générateur de CRC à 0. Si l'on sélectionne le mode SDLC, cette commande initialise le générateur de CRC à 1. Le vérificateur de CRC réception est aussi initialisé à 1 pour le mode SDLC.

Commande 6 (Initialisation d'erreur). Cette commande initialise les indicateurs d'erreur. Ces erreurs de parité et de surcharge sont verrouillées dans RR1 jusqu'à ce qu'elles soient remises à zéro par cette commande. De cette façon, les erreurs de priorité se produisant dans les transferts de blocs peuvent être examinées à la fin du bloc.

Commande 7 (Retour d'interruption). Cette commande doit être émise dans le canal A ; elle est interprétée par le SIO exactement de la même façon qu'une commande RETI dans le bus de données. Elle initialise le *latch* d'interruption en service du dispositif interne en service ayant la priorité la plus élevée, et permet ainsi aux dispositifs de plus basse priorité d'interrompre par le chaînage. Cette commande permet d'utiliser le chaînage interne même dans les systèmes sans chaînage ou commande RETI.

REGISTRE D'ECRITURE 1

WR1 contient les bits de commande des différents modes d'interruption et d'Attente/Prêt.

D_7 Attente/prêt Validation	D_6 $\overline{\text{Attente}}$ ou prêt Fonction	D_5 Attente/prêt Récep./Trans.	D_4 Réception Mode d'Inter. 1
D_3 Réception Interrup. Mode 0	D_2 Vecteur d'affectation d'état	D_1 Validation Interruption Transmission	D_0 Validation Interruptions externes

Validation d'interruption externe/état (D_0). La validation d'interruption externe/état permet aux interruptions d'avoir lieu à la suite de transitions sur les entrées $\overline{\text{DCD}}$, $\overline{\text{CTS}}$ ou $\overline{\text{SYNC}}$, à la suite d'une détection et d'une fin de Break/Abort, ou bien au début d'une transmission de CRC ou de caractère de sync quand le *latch* de transmission Surcharge/EOM est établi.

Validation d'interruptions de transmetteur (D_1). Si elles sont validées, les interruptions ont lieu chaque fois que le tampon du transmetteur se vide.

Etat affecte vecteur (D_2). Ce bit est actif dans le canal B seulement. Si ce bit n'est pas à 1, le vecteur fixe programmé dans WR2 est renvoyé d'une séquence d'accusé de réception d'interruption. S'il est à 1, le vecteur de retour d'un accusé de réception d'interruption est variable selon les conditions d'interruption suivantes :

	V_3	V_2	V_1	
Canal B	0	0	0	Canal B Tampon transmetteur vide
	0	0	1	Canal B Changement Externe/Etat
	0	1	0	Canal B Caractère réception disponible
	0	1	1	Canal B Condition réception spéciale *
Canal A	1	0	0	Canal A Tampon transmetteur vide
	1	0	1	Canal A Changement Externe/Etat
	1	1	0	Canal A Caractère réception disponible
	1	1	1	Canal A Condition réception spéciale *

* Conditions de réception spéciale, erreur de parité, erreur de surcharge Rx, erreur de cadrage, fin de cadrage (SDLC).

Modes 0 et 1 d'interruption de réception (D_3 et D_4). Ensemble, ces deux bits spécifient les différentes conditions de caractères disponibles. En mode 1, 2 et 3 d'interruption en réception, une condition de réception spéciale peut causer une interruption et modifier le vecteur d'interruption.

D_4 Réception Interruption Mode 1	D_3 Réception Interruption Mode 0	
0	0	0. Interruptions en réception inhibées
0	1	1. Interruptions en réception sur premier caractère seulement
1	0	2. Interruptions sur tous caractères reçus — erreur de parité en conditions de réception spéciales
1	1	3. Interruptions sur tous caractères reçus — erreur de parité en conditions de réception non spéciales

Sélection de fonction Attente/Prêt (D_5-D_7). Les fonctions Attente/Prêt sont sélectionnées par D_5, D_6 et D_7. Cette fonction est validée en établissant Validation Attente/Prêt (WR1,D_7). La fonction Prêt est sélectionnée en mettant D_6 à 1 (fonction Attente/Prêt). Si ce bit est à 1, les sorties $\overline{\text{ATTENTE/PRÊT}}$ basculent de haut à bas quand le SIO est prêt à transférer des données. La fonction Attente est sélectionnée en mettant D_6 à 0. Si ce bit est à 0, la sortie $\overline{\text{ATTENTE/PRÊT}}$ est à l'état drain ouvert et va au niveau bas quand elle est active.

Les fonctions Attente et Prêt peuvent être utilisées, soit en mode transmission, soit en mode réception, mais pas simultanément. Si D_5 (Attente/Prêt ou Réception/Transmission) est mis à 1, la fonction Attente/Prêt répond à la condition du tampon de réception (vide ou plein). Si D_5 est mis à 0, la fonction Attente/Prêt répond à la condition du tampon de transmission (vide ou plein).

Les états logiques de la sortie ATTENTE/PRÊT, lorsqu'elle est active ou inactive, dépendent de la combinaison des modes sélectionnés. Ce qui suit est un résumé de ces combinaisons :

	Si $D_7 = 0$	
	et $D_6 = 1$	et $D_6 = 0$
	$\overline{\text{PRÊT}}$ est haut	$\overline{\text{ATTENTE}}$ est flottant
	Si $D_7 = 1$	
	et $D_5 = 0$	et $D_5 = 1$
$\overline{\text{PRÊT}}$	est haut quand tampon de transmission est plein.	$\overline{\text{PRÊT}}$ est haut quand tampon de réception est vide.
$\overline{\text{ATTENTE}}$	est bas quand tampon de transmission est plein et qu'un port de données de SIO est sélectionné.	$\overline{\text{ATTENTE}}$ est bas quand tampon de réception est vide et port de données SIO sélectionné.
$\overline{\text{PRÊT}}$	est bas quand tampon de transmission est vide.	$\overline{\text{PRÊT}}$ est bas quand tampon de réception est plein.
$\overline{\text{ATTENTE}}$	est flottant quand tampon de transmission est vide.	$\overline{\text{ATTENTE}}$ est flottant quand tampon de réception est plein.

La transition haut vers le bas de la sortie ATTENTE a lieu avec le retard t_D IC(WR) après la demande d'entrée-sortie. La transition bas vers haut a lieu avec le retard t_D HΦ(WR) à partir du front descendant de Φ. La transition haut vers bas de la sortie PRÊT a lieu avec le retard t_D LΦ(WR) à partir du front montant de Φ. La transition bas vers haut de la sortie PRÊT a lieu avec le retard t_D IC(WR) après descente de $\overline{\text{IORQ}}$.

La fonction Prêt peut survenir chaque fois que le SIO n'est pas sélectionné. Quand la sortie PRÊT devient active (basse), le contrôleur de DMA émet $\overline{\text{IORQ}}$ et les signaux B/$\overline{\text{A}}$ et C/$\overline{\text{D}}$

correspondants vers le SIO pour transférer des données. La sortie PRÊT devient inactive aussitôt que $\overline{IORQ}$ et $\overline{CS}$ deviennent actifs. Comme la fonction Prêt peut avoir lieu à l'intérieur du SIO qu'il soit adressé ou non, la sortie PRÊT devient inactive lorsqu'un transfert de données de CPU ou une commande a lieu. Ceci ne cause pas de difficultés, car le contrôleur de DMA n'est pas validé quand le transfert de CPU a lieu.

La fonction Attente - par contre - est active seulement si le CPU essaie de lire les données du SIO qui n'ont pas encore été reçues, ce qui arrive fréquemment quand on utilise des instructions de transfert de bloc. La fonction Attente peut aussi devenir active (sous contrôle du programme) si le CPU essaie d'écrire des données pendant que le tampon est encore plein. Le fait que la sortie ATTENTE de l'un ou l'autre canal peut devenir active quand le canal opposé est adressé (parce que le SIO est adressé) n'affecte pas le fonctionnement des boucles logicielles ou des instructions de déplacement de blocs.

REGISTRE D'ECRITURE 2

WR2 est le registre de vecteur d'interruption. Il n'existe que dans le canal B. V_4-V_7 et V_0 sont toujours renvoyés exactement comme ils ont été écrits. V_1 à V_3 sont renvoyés comme écrits, si le bit de commande du vecteur d'affectation d'état (WR1, D_2) est à 0. Si ce bit est à 1, ils sont modifiés comme expliqué dans la section précédente.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0

REGISTRE D'ECRITURE 3

WR3 contient les paramètres et les bits de commande de logique de réception.

D_7	D_6	D_5	D_4
Bits récepteurs car. 1	Bits récepteurs car. 0	Auto valide	Phase recherche entier
D_3	D_2	D_1	D_0
Validation CRC récepteur	Mode recherche d'adresse	Inhib. charge car. de sync.	Valide réception

Validation de réception (D_0). Un 1 dans ce bit permet le début du fonctionnement en réception. Ce bit doit être mis à 1 seulement après que tous les paramètres aient été mis à 1 et que le récepteur ait été complètement initialisé.

Inhiber charge caractère sync (D_1). Les caractères de sync précédant le message (caractère de sync en tête) ne sont pas chargés dans les tampons de réception, si cette option est sélectionnée. Comme les calculs de CRC ne sont pas arrêtés par l'envoi de caractère de sync, ce caractère devrait être validé seulement au début du message.

Mode de recherche d'adresses (D_2). Si SDLC est sélectionné, l'établissement de ce mode provoque le rejet des messages des adresses ne correspondant pas aux adresses programmées en WR6 ou à l'adresse globale (11111111). En d'autres termes, aucune interruption de réception ne peut avoir lieu en mode de recherche d'adresse à moins qu'il n'y ait concordance d'adresses.

Validation CRC récepteur (D_3). Si ce bit est mis à 1, les calculs de CRC débutent ou repartent au commencement du dernier caractère transféré du registre à décalage de réception vers la pile du tampon, quel que soit le nombre de caractères dans la pile. Voir « Vérification de CRC de réception SDLC » (section réception SDLC) et « Vérification d'erreur de CRC » (section réception synchrone) pour les détails concernant l'établissement de ce bit.

Phase de recherche d'entrée (D_4). Le SIO entre automatiquement la phase de recherche après une initialisation. Elle peut cependant être réentrée si la synchronisation de caractère est perdue pour une raison quelconque (mode synchrone), ou si le contenu d'un message d'arrivée n'est pas nécessaire (mode SDLC). La phase de recherche est réentrée en écrivant un 1 dans le bit D_4. Ceci met le bit Sync/Recherche (D_4) dans RR0.

Auto-validation (D_5). Si ce mode est sélectionné, $\overline{DCD}$ et $\overline{CTS}$ deviennent les validations de récepteur et de transmetteur, respectivement. Si ce bit n'est pas mis à 1, $\overline{DCD}$ et $\overline{CTS}$ sont simplement envoyés dans leurs bits d'état correspondants dans RR0.

Bits/caractère 1 et 0 (D_7 et D_6) de récepteur. Ensemble, ces bits déterminent le nombre de bits de réception série assemblés pour former un caractère. Ces deux bits peuvent être changés pendant qu'un caractère est assemblé, mais avant que l'on ait atteint le nombre de bits programmés.

D_7	D_6	Bits/Caractères
0	0	5
0	1	7
1	0	6
1	1	8

REGISTRE D'ECRITURE 4

WR4 contient les bits de commande qui affectent à la fois le récepteur et le transmetteur. Dans la routine d'initialisation de transmission et d'émission, ces bits doivent être mis à 1 avant d'émettre WR1, WR3, WR5, WR6 et WR7.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Cadence d'horloge 1	Cadence d'horloge 0	Modes de sync. 1	Modes de sync. 0	Bits d'arrêt 1	Bits d'arrêt 0	Parité paire/impaire	Parité

Parité (D_0). Si ce bit est mis à 1, une position de bit additionnelle (en plus de celles spécifiées dans la commande bits/caractère) est ajoutée aux données transférées et est attendue dans les données de réception. Dans le mode réception, le bit de parité reçu est transféré vers le CPU comme partie de caractère, à moins qu'un caractère 8 bits ne soit sélectionné.

Parité paire/$\overline{\text{impaire}}$ (D_1). Si la parité est spécifiée, ce bit détermine s'il est envoyé et vérifié comme pair ou impair (1 = pair).

Bits d'arrêt 0 et 1 (D_2 et D_3). Ces bits déterminent le nombre de bits d'arrêt ajoutés à chaque caractère asynchrone envoyé. Le récepteur vérifie toujours la présence d'un bit d'arrêt. Un mode spécial (00) spécifie qu'un mode synchrone doit être sélectionné.

D_3 Bits d'arrêt 1	D_2 Bits d'arrêt 0	
0	0	Modes sync.
0	1	1 bit d'arrêt par caractère
1	0	1 1/2 bit d'arrêt par caractère
1	1	2 bits d'arrêt par caractère

Mode sync 0 et 1 (D_4 et D_5). Ces bits déterminent les différentes options de synchronisation de caractère.

Sync Mode 1	Sync Mode 0	
0	0	Sync à 8 bits programmés
0	1	Sync à 16 bits programmés
1	0	Mode SDLC (01111110)
1	1	Mode Sync externe

Vitesse d'horloge 0 et 1 (D_6 et D_7). Ces bits spécifient le coefficient de multiplication entre l'horloge (TxC et RxC) et le taux de données. Pour les modes synchrones, le taux d'horloge × 1 doit être spécifié. Tous les taux peuvent être spécifiés pour des modes asynchrones, cependant le même taux doit être employé pour le récepteur et le transmetteur. Dans tous les modes, l'horloge du système doit être au moins 5 fois la vitesse des données. Si l'on sélectionne le facteur 1, la synchronisation doit être effectuée extérieurement.

Fonction d'horloge 1	Fonction d'horloge 0	
0	0	Taux de données × 1 = Taux d'horloge
0	1	Taux de données × 16 = Taux d'horloge
1	0	Taux de données × 32 = Taux d'horloge
1	1	Taux de données × 64 = Taux d'horloge

REGISTRE D'ECRITURE 5

WR5 contient les bits de commande qui affectent le fonctionnement du transmetteur, à l'exception de D2 qui affecte le transmetteur et le récepteur.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
DTR	Tx Bits/ Caract. 1	Tx Bits/ Caract. 0	Envoi Arrêt	Validat. Tx	CRC-16 SDLC	RTS	Validat. Tx CRC

Validation CRC transmetteur (D_0). Ce bit détermine si CRC est calculé sur un caractère de transmission particulier. S'il est mis à 1 au moment où le caractère est chargé du tampon de transmission dans le registre à décalage du transmetteur, CRC est calculé sur le caractère. CRC n'est pas envoyé automatiquement à moins que ce bit ne soit mis à 1 quand existe une condition de surcharge du transmetteur.

Demande d'envoi (D_1). C'est le bit de commande pour la broche $\overline{RTS}$. Quand le bit $\overline{RTS}$ est à 1, la broche $\overline{RTS}$ va au niveau bas ; quand on l'initialise $\overline{RTS}$ va au niveau haut. En mode asynchrone, $\overline{RTS}$ va au niveau haut seulement après transmission des bits du caractère et une fois le tampon du transmetteur vide. Dans les modes synchrones, la broche suit directement l'état du bit.

CRC-16/$\overline{SDLC}$ (D_2). Ce bit sélectionne le polynôme CRC utilisé pour le transmetteur et le récepteur. Quand il est à 1, le polynôme CRC-16 ($X^{16} + X^{15} + X^2 + 1$) est utilisé ; quand il est mis à 0, le polynôme SDLC ($X^{16} + X^{12} + X^{15} + 1$) est utilisé. Si le mode SDLC est sélectionné, le générateur CRC et le vérificateur sont remis à 1, et une séquence de vérification spéciale est employée. Le polynôme CRC SDLC doit être sélectionné quand le mode SDLC l'est. Si le mode SDLC n'est pas sélectionné, le générateur de CRC et le vérificateur sont présents pour tous les 0 (pour les deux polynômes).

Validation de transmission (D_3). Les données ne sont pas transmises jusqu'à ce que ce bit soit mis à 1 et que la sortie de données de transmission soit maintenue en marquage. Les données ou les caractères de sync en cours de transmission sont complètement envoyés si ce bit est remis à zéro après le début de la transmission. Si le transmetteur est inhibé, des caractères de sync ou de drapeau sont envoyés au lieu de CRC.

Envoyer Break (D_4). Quand il est mis à 1, ce bit force immédiatement la sortie de données de transmission dans la condition espacement, quelles que soient les données transmises. Quand il est remis à 0, TxD retourne au marquage.

Transmission Bits/Caractère 0 et 1 (D_5 et D_6). Ensemble, D_5 et D_6 commandent le nombre de bits de chaque octet transféré au tampon de transmission.

D_6 Bits de transm. Caractère 1	D_5 Bits de transm. Caractère 0	Bits/Caractère
0	0	Cinq ou moins
0	1	7
1	0	6
1	1	8

Les bits à envoyer doivent être justifiés à droite, bits de poids faible d'abord. Le mode 5 ou moins permet la transmission de un à cinq bits par caractère ; cependant le CPU doit assembler les caractères de données comme le montre le tableau.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	1	1	1	0	0	0	D	Envoie un bit de données
1	1	1	0	0	0	D	D	Envoie deux bits de données
1	1	0	0	0	D	D	D	Envoie trois bits de données
1	0	0	0	D	D	D	D	Envoie quatre bits de données
0	0	0	D	D	D	D	D	Envoie cinq bits de données

Terminal de données prêt (D_7). C'est le bit de commande pour la broche $\overline{DTR}$. Quand il est mis à 1, $\overline{DTR}$ est actif (bas) ; quand il est remis à 0, $\overline{DTR}$ est inactif (haut).

REGISTRE D'ECRITURE 6

Ce registre est programmé pour contenir le caractère de sync de transmission en mode monosync, les huit premiers bits d'un caractère de sync à 16 bits en mode bisync, ou un caractère de sync de transmission en mode de sync externe. En mode SDLC il est programmé pour contenir le champ d'adresse secondaire utilisé pour comparer avec le champ d'adresses du cadre SDLC.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Sync 7	Sync 6	Sync 5	Sync 4	Sync 3	Sync 2	Sync 1	Sync 0

REGISTRE D'ECRITURE 7

Ce registre est programmé pour contenir le caractère de sync de réception en mode monosync, un second octet (les 8 derniers bits) d'un caractère de sync à 16 bits en mode bisync et un caractère drapeau (01111110) en mode SDLC. WR7 n'est pas employé en mode sync externe.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Sync 15	Sync 14	Sync 13	Sync 12	Sync 11	Sync 10	Sync 9	Sync 8

INTRODUCTION AUX REGISTRES DE LECTURE

Le SIO contient trois registres, RR0-RR2 (Figure 7.1), qui peuvent être lus pour obtenir l'information d'état de chaque canal (excepté canal B de RR2). L'information d'état comporte les conditions d'erreur, le vecteur d'interruption et les signaux standard de communications interfaces.

Pour lire le contenu d'un registre de lecture sélectionné autre que RR0, le programme du système doit d'abord écrire l'octet pointeur dans WR0 exactement de la même manière qu'une opération d'écriture en registre. Puis, par exécution d'une instruction d'entrée, le contenu du registre de lecture adressé peut être lu par le CPU.

Les bits d'état de RR0 et RR1 sont soigneusement groupés pour simplifier la surveillance des états. Par exemple, quand le vecteur d'interruption indique qu'une interruption de condition de réception spéciale a eu lieu, tous les bits d'erreur appropriés peuvent être lus dans un seul registre distinct (RR1).

REGISTRE DE LECTURE 0

Ce registre contient l'état des tampons de réception et de transmission, les entrées $\overline{DCD}$, $\overline{CTS}$ et $\overline{SYNC}$, le *latch* de transmission Underrun/EOM et le *latch* Break/Abort.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Break Abort	Transmis. Underrun/ EDM	CTS	Sync/ rech.	DCD	Transmis. tampon vide	Interrup. (canal A seulement)	Réception caractère disponible

Caractère de réception disponible (D_0). Ce bit est mis à 1 quand un caractère au moins est disponible dans le tampon de réception : il est mis à 0 quand le FIFO de réception est complètement vide.

Interruption imminente (D_1). Dans le SIO, toute condition d'interruption met ce bit à 1 ; il ne peut, cependant, être lu que dans le canal A. Ce bit est principalement utilisé dans les applications qui n'ont pas d'interruptions vectorisées. Ceci élimine la nécessité d'analyser tous les bits de RR0 dans les canaux A et B. Le bit D_1 est remis à 0 quand toutes les conditions d'interruption sont satisfaites. Ce bit est toujours à 0 dans le canal B.

Tampon de transmission vide (D_2). Ce bit est mis à 1 chaque fois que le tampon de transmission se vide, excepté quand un caractère CRC est envoyé en mode synchrone ou SDLC. Le bit est remis à 0 quand un caractère est chargé dans le tampon de transmission. Ce bit est à 1 après un reset.

Détection de porteur de données (D_3). Ce bit DCD montre l'état inversé de l'entrée $\overline{DCD}$ au moment du dernier changement d'un des cinq bits Externe/Etat (DCD, $\overline{CTS}$, Sync/Hunt, Break/Abort ou Underrun/EOM). Toute transition de l'entrée $\overline{DCD}$ fait *latcher* le bit DCD et cause une interruption Externe/Etat. Pour lire l'état en cours de bit DCD, ce bit doit être lu immédiatement après une commande d'interruption Externe/Etat d'initialisation.

Sync/Hunt (D_4). Comme ce bit est commandé différemment en mode asynchrone, synchrone et SDLC, son fonctionnement est un peu plus complexe que celui des autres bits et nécessite, par conséquent, davantage d'explications.

Dans les modes asynchrones, le fonctionnement de ce bit est similaire à celui du bit d'état DCD, excepté que Sync/Hunt montre l'état de l'entrée $\overline{SYNC}$. Toute transition haut vers bas de la broche $\overline{SYNC}$ met le bit à 1 et provoque une interruption Externe/Etat (si elle est

validée). La commande d'interruption de remise à 0 Externe/Etat est émise pour remettre à 0 l'interruption. Une transition bas vers haut remet ce bit à 0 et met à 1 l'interruption Externe/Etat. Quand l'interruption Externe/Etat est mise à 1 par le changement d'état de toute autre entrée ou condition, ce bit montre l'état inversé de la broche $\overline{SYNC}$ au moment du changement. Ce bit doit être lu immédiatement après une commande d'interruption de remise à 0 Externe/Etat pour lire l'état en cours de l'entrée $\overline{SYNC}$.

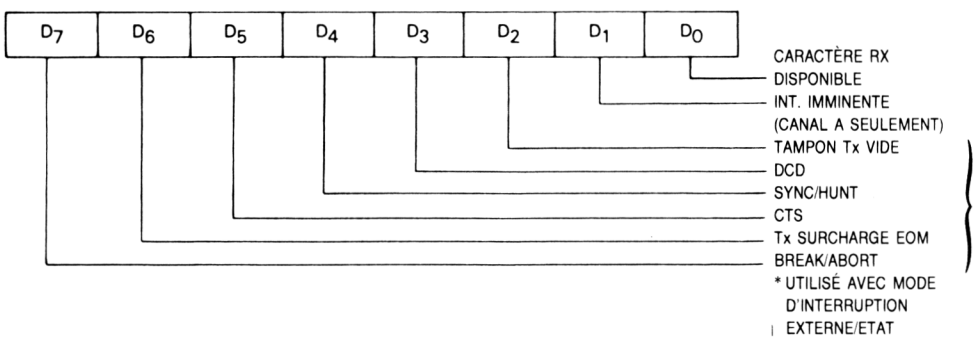
Dans le mode Sync externe, le bit Sync/Recherche fonctionne d'une façon similaire au mode asynchrone, sauf que le bit de commande du mode Enter Hunt valide la logique de détection de sync externe. Quand les bits de mode sync externe et de mode Enter Hunt sont mis à 1 (par exemple quand le récepteur est validé à la suite d'une remise à 0), l'entrée $\overline{SYNC}$ doit être maintenue au niveau haut par la logique externe jusqu'à ce qu'une synchronisation de caractère externe soit obtenue. Un niveau haut à l'entrée $\overline{SYNC}$ maintient le bit d'état Sync/Haut en condition de remise à 0.

Quand une synchronisation externe est obtenue, $\overline{SYNC}$ doit être entraîné au niveau bas au second front montant de $\overline{RxC}$ sur lequel le dernier bit du caractère sync a été reçu. En d'autres termes, après détection de la configuration Sync, la logique externe doit attendre 2 cycles d'horloge complets de réception pour activer l'entrée $\overline{SYNC}$. Une fois $\overline{SYNC}$ forcé au niveau bas, c'est une bonne habitude de le laisser au niveau bas jusqu'à ce que le CPU informe la logique de sync externe que la synchronisation a été perdue ou qu'un nouveau message est sur le point d'être lancé. Voir la Figure 8.6 pour les détails de synchronisation. La transition haut vers bas de l'entrée $\overline{SYNC}$ met à 1 le bit Sync/Hunt, qui à son tour met à 1 l'interruption Externe/Etat. Le CPU doit remettre à 0 l'interruption en émettant la commande d'interruption de remise à 0 Externe/Etat.

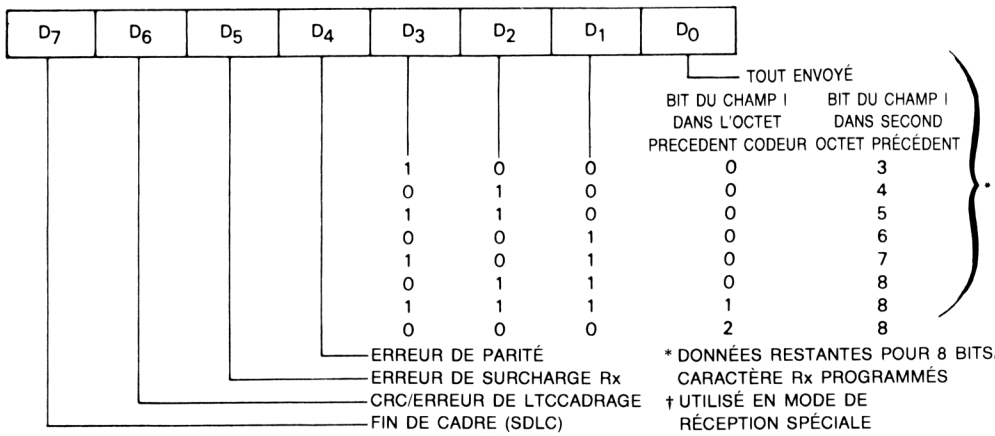
Quand l'entrée $\overline{SYNC}$ retourne au niveau haut, une autre interruption Externe/Etat est produite et doit être aussi remise à 0. Le bit de commande de mode Enter Hunt est mis à 1 chaque fois qu'une synchronisation de caractère est perdue ou que la fin d'un message est détectée. Dans ce cas, le SIO cherche à nouveau une transition haut vers bas sur l'entrée $\overline{SYNC}$ et l'opération se répète comme expliqué précédemment. Ceci implique que le CPU doit aussi informer la logique externe qu'une synchronisation de caractère a été perdue et que le SIO attend que $\overline{SYNC}$ devienne actif.

FONCTIONS DES BITS DU REGISTRE DE LECTURE

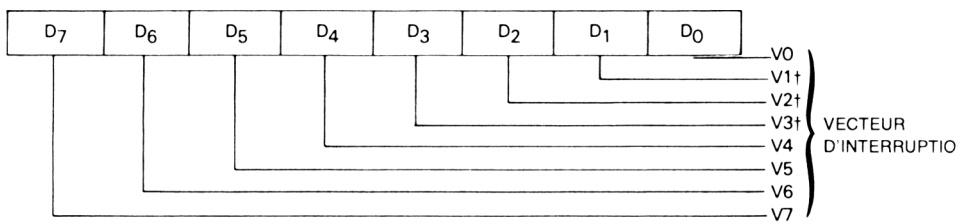
REGISTRE DE LECTURE 0



REGISTRE DE LECTURE 1†



REGISTRE DE LECTURE 2



Dans les modes de réception monosync et bisync le bit d'état Sync/Hunt est mis initialement à 1 par bit de mode Enter Hunt. Le bit Sync/Hunt est remis à 0 quand le SIO établit la synchronisation de caractères. La transition haut vers bas du bit Sync/Hunt cause une interruption Externe/Etat qui doit être remise à 0 par le CPU, quand il émet la commande d'interruption de remise à zéro Externe/Etat. Ceci permet au SIO de détecter la transition suivante d'autres bits Externe/Etat.

Quand le CPU détecte la fin de message de ce caractère et que la synchronisation est perdue, il met à 1 le bit de commande en mode Enter Hunt, lequel à son tour met à 1 le bit Sync/Hunt. La transition bas vers haut du bit Sync/Hunt établit l'interruption Externe/Etat qui doit être aussi mise à 0 par la commande d'interruption de remise à 0 Externe/Etat. Remarquez que la broche Sync agit en sortie dans ce mode et va au niveau bas chaque fois qu'une configuration de sync est détectée dans le flot de données.

Dans le mode SDLC, le bit Sync/Hunt est initialisé par le bit de mode Enter Hunt, ou quand le récepteur est inhibé. Dans tous les cas, il est remis à 0 quand le drapeau d'ouverture du premier cadre est détecté par le SIO. L'interruption Externe/Etat est produite et doit être traitée comme indiqué précédemment.

Contrairement aux modes monosync et bisync, une fois que le bit Sync/Hunt est remis en mode SDLC, il n'est pas nécessaire de le rétablir quand la fin de message est détectée. Le SIO maintient automatiquement la synchronisation. La seule manière d'établir à nouveau le bit Sync/Hunt est d'employer le bit de mode Enter Hunt ou bien d'inhiber le récepteur.

Mettre à 0 pour envoyer (D_5). Ce bit est semblable au bit DCD excepté qu'il montre l'état inversé de la broche $\overline{CTS}$.

Transmission Underrun/Fin de message (D_6). Ce bit est dans une condition de mise à 1 suivant une réinitialisation interne ou externe. La seule commande, pouvant remettre ce bit à 0, est la commande de *latch* Underrun/EM de remise à 0 de transmission (D_6 et D_7 de WR0). Quand la condition Underrun de transmission a lieu, ce bit est mis à 1, ce qui cause l'interruption Externe/Etat qui doit être remise à 0 en envoyant les bits de commande d'interruption de remise à 0 Externe/Etat (WR0). Ce bit d'état joue un rôle important, en liaison avec d'autres bits de commande, pour contrôler une opération de transmission. Se référer à *Bisync Transmit Underrun* et *SDLC Transmit Underrun* pour des détails supplémentaires.

Break/Abort (D_7). Dans le mode de réception asynchrone, ce bit est mis à 1 quand une séquence Break (caractère nul, plus erreur de cadrage) est détectée dans le flot de données. L'interruption Externe/Etat, si elle est validée, est mise à 1 quand Break est détecté. La routine de service d'interruption doit envoyer la commande d'interruption de remise à 0 Externe/Etat (WR0, CMD_2) à la logique de détection de Break, pour que la fin de séquence de Break soit reconnue.

Le bit Break/Abort est mis à 0 quand la fin de la séquence de Break est détectée dans le flot de données d'entrée. La fin de la séquence de Break provoque aussi la mise à 1 de l'interruption Externe/Etat. La commande d'interruption Externe/Etat de remise à 0 doit être envoyée pour valider la logique de détection de Break pour chercher la séquence de Break suivante. Si un caractère nul est présent dans le récepteur après la fin d'un Break, il doit être lu et écarté.

Dans le mode de réception SDLC, le bit d'état est mis à 1 par la détection d'une séquence Abort (7 « 1 » ou plus). L'interruption Externe/Etat est manipulée de la même façon que dans le cas d'un Break. Le bit Break/Abort n'est pas employé en mode de réception synchrone.

REGISTRE DE LECTURE 1

Ce registre contient les bits d'état de condition de réception spéciale et les codes résiduels pour le champ I en mode de réception SDLC.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Fin de cadre (SDLC)	Erreur CRC/ cadrage	Erreur de surcharge de réception	Erreur de parité	Code 2 résiduel	Code 1 résiduel	Code 0 résiduel	Tout envoyé

Tout envoyé (D_0). Dans les modes asynchrones, ce bit est mis à 1 quand tous les caractères ont complètement mis à 0 le transmetteur. Les transitions de ce bit ne causent pas d'interruption. Ce bit est toujours mis à 1 dans les modes synchrones.

Codes résiduels 0, 1, et 2 (D_1-D_3). Dans les modes de réception SDLC où le champ I n'est pas un multiple entier de la longueur du caractère, ces trois bits indiquent la longueur du champ I. Ces codes n'ont de sens que pour les transferts dans lesquels le bit de fin de cadre est mis à 1 (SDLC). Pour une longueur de caractère de réception de huit bits par caractère, les codes ont les significatons suivantes :

Code 2 résiduel	Code 1 résiduel	Code 0 résiduel	Bits de champ I dans octet précédent	Bits de champ I dans second octet précédent
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8

Les bits du champ I sont justifiés à droite dans tous les cas.

Si une longueur de caractère de réception différente de huit bits est employée pour le champ I, une table semblable à la précédente peut être construite pour chaque longueur de caractère différente. Pour ne pas avoir de reste, c'est-à-dire que la limite du dernier caractère coïncide avec celle du champ I et du champ CRC, les codes résiduels sont :

Bits par caractère	Code 2 résiduel	Code 1 résiduel	Code 0 résiduel
8 Bits par caractère	0	1	1
7 Bits par caractère	0	0	0
6 Bits par caractère	0	1	0
5 Bits par caractère	0	0	1

Erreur de parité (D_4). Quand la parité est validée, ce bit est mis à 1 pour les caractères dont la parité ne coïncide pas avec la programmation (pair/impair). Ce bit est *latché* de sorte qu'en cas d'erreur, il reste mis à 1 jusqu'à la commande de remise à 0 d'erreur (WR0).

Erreur surcharge en réception (D_5). Ce bit indique que plus de trois caractères ont été reçus sans lecture du CPU. Seul le caractère recouvert en écriture fait l'objet d'un indicateur d'erreur, mais quand il est lu, la condition d'erreur est *latchée* jusqu'à remise à 0 par la commande de remise à 0 d'erreur. Si Etat Affecté Vecteur est validé, le caractère surchargé interrompt avec un vecteur de condition de réception spéciale.

Erreur CRC/Cadrage (D_6). Si une erreur de cadrage a lieu (mode synchrone), ce bit est mis à 1 (et non pas *latché*) par le caractère reçu dans lequel l'erreur de cadrage a eu lieu. La détection d'une erreur de cadrage ajoute un temps supplémentaire d'un demi-bit au temps de caractère, de sorte que l'erreur de cadrage n'est pas interprétée comme nouveau bit de départ. Dans les modes synchrones et SDLC, ce bit indique le résultat de comparaison entre vérificateur de CRC et valeur de vérification appropriée. Ce bit est remis à 0 en envoyant une commande de remise à 0 d'erreur. Ce bit n'est pas *latché* et est donc remis à jour quand le caractère suivant est reçu. Lorsqu'il est utilisé pour des erreurs de CRC et d'état en mode synchrone, il est habituellement mis à 1, puisque la plupart des combinaisons de bits produisent un CRC non nul, excepté pour un message correctement complété.

Fin de format (D_7). Ce bit sert seulement en mode SDLC, et indique qu'un drapeau de fin valide a été reçu et que l'erreur de CRC et les codes résiduels sont aussi valides. Ce bit peut être remis à 0 en envoyant la commande de remise à 0 d'erreur. Il est aussi renouvelé par le premier caractère du format suivant.

REGISTRE DE LECTURE 2 (canal B seulement)

Ce registre contient le vecteur d'interruption écrit dans WR2, si le bit de commande Etat Affecte Vecteur n'est pas mis à 1. Si le bit de commande est mis à 1, il contient le vecteur de la section du registre d'écriture 1. Quand ce registre est lu, le vecteur retourné est modifié par la condition d'interruption de priorité supérieure au moment de la lecture. S'il n'y a pas d'interruptions en attente, le vecteur est modifié avec $V_3 = 0$, $V_2 = 1$ et $V_1 = 1$. Ce registre ne peut être lu que par l'intermédiaire du canal B.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0
					Variable si Etat Affecte Vecteur est validé		

INDEX

Adressage,
de lignes d'adresses, 3, 4.
de ROM plus grandes, 16.
Adresse,
Calculs d', 3.
Connexions à RAM, 37-41.
Connexions d', 37.
Ligne d', 7.
Mapping d', 6, 8.
Tampon d', 18, 19.
Applications du 8251, 245-249.
Architecture, 56, 57.
E/S mappée, 56, 57.
Sélection linéaire, 56.
Arrêt,
Bit d'(définition), 227.

Bit,
d'arrêt, 215, 227.
de départ, 225.
de parité, 225.
Brochage,
du 8251, 232.
du Z80-CTC, 201.
du Z80-PIO, 172-177.
du Z80-SIO, 254-259.
Broche de déclenchement du 8253, 166-168.

Canal,
Bloc de (Z80-CTC), 200.
Programmation de, 209.
Registre de commande de, 210.
CAS (Column Address Strobe), 74.
Génération de, 80, 81.
Chip Select, 4, 10-13.
Clavier, 134.
matrice, 134.
programme d'exploration, 135.
CMS (Creative Microprocessor System), 289.

Commande
d'écriture en mémoire, 42.
de lecture en mémoire, 42.
IOR, 59, 60.
IOW, 59, 60.
Ligne de, 42.
Mot de, 128.
Communications,
Série, 221, 229.
Compteur-temporisateur, 201.
brochage, 201.
Connexions aux bus du Z80, 234.
en compteur général, 216.
en mode compteur, 207.
Fonctionnement général de temporisateur, 216.
Schéma de boîtier, 202.
Configuration,
Mode 0, 131.
Conflit de bus, 35.
Connexion,
de lignes d'adresses et de données à une ROM, 7.
de ports du système, 7, 151-154.
des bus du Z80 au temporisateur 8253, 151-154.
du CTC aux bus du Z80, 204.
du 8251 aux bus du Z80, 234.
du PIO aux bus du Z80, 177.
du SIO aux bus du Z80, 259.
Conversion,
série-parallèle, 224.

Départ,
Bit de (définition), 225.
Diagramme des temps, 30.
pour une écriture en RAM, 31.
pour une lecture en RAM, 30.
Données,
amplifiées, 21.

d'entrée en RAM dynamique, 81.
de sortie, en série, 235.
Feuille de, 33.
Lignes de, 41.
non amplifiées, 41.
Dynamique, RAM, 71-91.
Aperçu général, 71.
Ecriture de données, 86.
Entrée de données, 81.
Lecture de données, 86.
Multiplexage des lignes d'adresses, 76-78.
Rafraîchissement, 88-91.
Schéma complet d'un système, 90.
Schéma de système, 79.

EAROM, définition, 3.
Ecriture,
dans la RAM 2114, 36.
de données dans une RAM dynamique, 83.
registre, 127.
Entrées et sorties,
Aperçu d', 55.
communes, 32.
séparées pour RAM, 32.
EPROM, définition, 2, 3.
Erreur, 245.
Format, 245.
Surcharge, 245.
E/S,
Aperçu, 55.
Architecture, 55.
Schéma de port, 65.
séparées pour RAM, 32.

Handshaking, 137.

INT, entrée, 101.
Interruption,
à chaînage de priorité, 118.
à demandes multiples, 114.
Définition, 93.
Diagramme des entrées, 95.
Entrée INT, 101.
Mode 0, 105.
Mode 1, 102.
Mode 2, 109.
non masquable, 95.
Priorité, 115.
Scrutation. 115.
Source, 94.
Vecteur d' (CTC), 211.
Interruptions non masquables (NMI), 95.
Exemples d', 98.
Remise à 0 d', 98.
Résumé d', 101.
IOR,
Ligne de commande, 59.
IOW,
Ligne de commande, 59.

Lecture,
de données, 6.
de la RAM 2114, 37.
d'une RAM dynamique, 86.
d'une ROM (diagramme des temps), 6.
Registre, 149.
Temps d'accès (définition), 29, 31.
Lecture d'entrée,
Suite des événements pour, 5, 6.
Ligne,
Driver, 238.
Récepteur, 239.
Lignes de données pour RAM amplifiées, 45.

MC1488, émetteur de ligne, 238.
MC1489, récepteur de ligne, 239.
Mémoire,
Ligne de commande d'écriture, 42.
Ligne de commande de lecture, 42.
Production de sélection, 41.
Production de signaux de lecture, 11.
Mode 0,
Exemple de 8255, 131.
Exemple de temporisateur ou compteur 8253, 158.
Interruptions, 105.
8255, 131.

Mode 1,
Exemple de temporisateur ou compteur 8253, 161.
Interruptions, 102.
Mode 2,
Exemple de temporisateur 8253, 161.
8255, 143.
Interruptions, 109.
Mode 3,
Exemple de temporisateur 8253, 163.
Mode 4,
Exemple de temporisateur 8253, 165.
Mode 5,
Exemple de temporisateur 8253, 166.
Mot,
de commande pour le 8255, 128.
Multiples,
Demande d'interruptions, 114.
Multiplexage, 76.
d'adresses pour RAM dynamique, 76.
MUX,
Génération de signal, 80.

NMI (voir Interruptions non masquables)

Parité,
Bit de (définition), 225.
PIO, 8255, 121.
Connexion aux bus du Z80, 125.
Mode 0, 129.
Registres de lecture-écriture, 127.
Port,
Adresses, 56.
Ligne de sélection, 57, 58.
Signal d'écriture, 62.
Signal de lecture, 62.
Priorité,
en daisy chain, 118.
interruptions, 115.
Production du signal de lecture en mémoire, 11.
Programmable (timer 8253),
Schéma, 148.
Temporisation (8253), 148.
Programmation,
Exemples de (Z80-CTC), 212.
du 8251, 237.
du 8253, 154.
Programme de saisie de clavier, 135.
PROM, Définition, 2, 3.

Rafraîchissement,
de la RAM dynamique, 88.
RAM, (Random Access Memory),
Aperçu du composant statique, 27-29.
Boîtier 2114, 32
Boîtier 6116 2 K × 8, 45-51.
Calcul du nombre de lignes d'adresses, 37.
Connexions des lignes d'adresses, 37-41.
Connexions des lignes de données, 41-42.
Définition, 27.
Diagramme des temps d'une opération d'écriture, 31.
Diagramme des temps d'une opération de lecture, 29, 30.
Entrées et sorties communes, 32, 33.
E/S séparées, 33.
Feuille de données de la 2114, 33.
Schéma de composant statique, 30.
Fonctionnement en écriture, 31.
Schéma d'une 2 K × 8, 46.
Schéma d'une 4 K × 8, 51.
Suite d'événements de lecture, 29, 30.
Temps d'accès en écriture, 31.
RAS, (Row Address Strobe), 74.
Génération de, 80, 81.
Réception,
de données en série (Z80-SIO), 265.
Registres internes (8253), 149-150.
Remise à 0 du Z80-PIO, 178.

ROM (Read Only Memory),
Augmentation, 15.
Caractéristiques de synchronisation, 3-5.
Connexion d'adresses et de données, 7.
2732 et 2764, 17, 18.
Définition, 1, 2.
Diagramme des temps en lecture de données, 5, 6.
Exemples de systèmes, 25.
Organisation, 3, 4.
Schéma de Z80 connecté à, 12.
Schéma, 4.
Suite d'opérations de lecture de données, 6.
Row Address Strobe (voir RAS), 74, 80.
Schéma,
d'adresses et de données connectées à une ROM, 7.
de la RAM 2114, 90.
de RAM statique, 46.
du 8253, 147.
du Z80-CTC, 199.
du Z80-PIO, 171.
du Z80-SIO, 253.
d'un port E/S, 64, 65.
d'un système à RAM dynamique, 78, 79.
d'un Z80 connecté à une ROM, 7.
d'une RAM 2 K × 8 statique, 51.
d'une RAM 4 K × 8 statique, 46.
Scrutation, 115.
Sélection de boîtier, 4, 10, 11.
Définition, 4.
Production, 10, 11.
Série,
Conversions de données série-parallèle, 224.
Dispositif (8251), 229.
Données de sortie, 229.
Introduction à communication, 221-227.
Timing, 222.
SST (Static Stimulus Testing), 279-289.
Aperçu, 280-283.
Circuit, 283-285.
Statique,
RAM (voir RAM)
Strobe,
Ecriture de port, 62.
Lecture de port, 62.
Suite d'événements,
pour sortie d'écriture, 64, 67.
pour une écriture de données 2114, 36.
pour une écriture en RAM, 31.
pour une écriture en sortie, 64-66.
pour une lecture de données 2114, 37.
pour une lecture d'entrée, 68.
pour une lecture en RAM, 29, 30.
Système,
ROM, 25.
Tampons, 18-24.
d'adresses, 18-20.
de données, 21-24.
Temporisateur (Timer), 147-168.
Broche d'entrée de déclenchement, 166-168.
Connexion aux bus du Z80, 151-154.
Exemple de mode 0, 158-161.
Exemple de mode 1, 161.
Exemple de mode 2, 161.
Exemple de mode 3, 163.
Exemple de mode 4, 165.
Exemple de mode 5, 166.
8253, 147.
programmable, 147.
Programmation de, 154-158.
Temporisation,
série, 222-223.
Temps,
Constante de (Z80-CTC), 211.
Programmation de registre de, 211.

Temps d'accès, 31, 29.
en écriture, 31.
en lecture, 29.
Trois états,
Définition, 4, 5.
Z80-CTC, 199-218.
Brochage, 201.
Canaux, 200.
Connexion aux bus du Z80, 204.
Exemples de programmation, 212.
Fonctionnement général de compteur, 216-217.
Fonctionnement général du temporisateur, 212-215.
Mode compteur, 207-209.
Programmation du registre de commande des canaux, 209-210.
Programmation du registre de constante de temps, 211.
Schéma, 199.
Vecteurs d'interruption, 211, 212.
Z80-PIO,
Brochage, 172-177.
Connexion aux bus du Z80, 177.
Fonctionnement en mode 2, 188.
Fonctionnement en mode 3, 191.
Priorités d'interruption, 195-196.
Programmation, 179-183.
Récapitulation des modes 1 et 0, 187.
Remise à 0, 178.
Schéma, 171.
Validation et inhibition d'interruptions, 195.
Z80-SIO,
Brochage, 254.
Connexion aux bus du Z80, 259.
Exemple, 267.
Interruptions, 271.
Réception de données en série, 265.
Registres, 262.
Schéma, 253.
Séquence générale d'initialisation, 263.

LA BIBLIOTHÈQUE SYBEX

OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par Rodnay Zaks,*
296 pages, Réf. 394

VOTRE ORDINATEUR ET VOUS *par Rodnay Zaks,*
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par Rodnay Zaks,*
636 pages, Réf. 340

TECHNIQUES D'INTERFACE aux microprocesseurs *par Austin Lesea et Rodnay Zaks,*
450 pages, Réf. 339

LEXIQUE INTERNATIONAL MICRO-ORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

GUIDE DES MICRO-ORDINATEURS A MOINS 3 000 F *par Joël Poncet,*
144 pages, Réf. 322

LEXIQUE MICRO-INFORMATIQUE *par Pierre Le Beux,*
140 pages, Réf. 369

LA SOLUTION RS-232 *par Joe Campbell,*
208 pages, Réf. 0052

MINITEL ET MICRO-ORDINATEUR *par Pierrick Bourgault,*
198 pages, Réf. 0119

BASIC

VOTRE PREMIER PROGRAMME BASIC *par Rodnay Zaks,*
208 pages, Réf. 263

INTRODUCTION AU BASIC *par Pierre Le Beux,*
336 pages, Réf. 0035

LE BASIC PAR LA PRATIQUE : 60 exercices *par Jean-Pierre Lamoitier,*
252 pages, Réf. 0095

LE BASIC POUR L'ENTREPRISE *par Xuan Tung Bui,*
204 pages, Réf. 253

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par Alan R. Miller,*
318 pages, Réf. 259

BASIC, PROGRAMMATION STRUCTURÉE *par Richard Mateosian,*
352 pages, Réf. 429

JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
192 pages, Réf. 246

NOUVEAUX JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
204 pages, Réf. 247

FICHIERS EN BASIC *par Alan Simpson,*
256 pages, Réf. 0102

PASCAL

INTRODUCTION AU PASCAL *par Pierre Le Beux,*
496 pages, Réf. 330

LE PASCAL PAR LA PRATIQUE *par Pierre Le Beux et Henri Tavernier,*
562 pages, Réf. 361

LE GUIDE DU PASCAL *par Jacques Tiberghien,*
504 pages, Réf. 423

PROGRAMMES EN PASCAL pour Scientifiques et Ingénieurs *par Alan R. Miller,*
392 pages, Réf. 240

AUTRES LANGAGES

INTRODUCTION A ADA *par Pierre Le Beux,*
366 pages, Réf. 360

MICRO-ORDINATEURS

ALICE

JEUX EN BASIC POUR ALICE *par Pierre Monsaut,*
96 pages, Réf. 320

ALICE et ALICE 90, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 376

ALICE, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 401

ALICE, GUIDE DE L'UTILISATEUR *par Norbert Rimoux,*
208 pages, Réf. 378

ALICE, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot-Barraly,*
192 pages, Réf. 420

AMSTRAD

AMSTRAD, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 0105

AMSTRAD, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0107

AMSTRAD, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0108

AMSTRAD, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot-Barraly,*
208 pages, Réf. 0136

AMSTRAD EXPLORÉ *par John Braga,*
192 pages, Réf. 0135

APPLE / MACINTOSH

PROGRAMMEZ EN BASIC SUR APPLE II, Tomes 1 et 2 *par Léopold Laurent,*
208 pages, Réf. 333 et 380

APPLE II 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE
par Douglas Hergert et Joseph T. Kalash,
372 pages, Réf. 241

GUIDE DU BASIC APPLE II *par Douglas Hergert,*
272 pages, Réf. 0006

APPLE II, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 373

MACINTOSH, GUIDE DE L'UTILISATEUR
par Joseph Caggiano,
208 pages, Réf. 396

APPLE IIC, GUIDE DE L'UTILISATEUR
par Thomas Blackadar,
160 pages, Réf. 0089

MULTIPLAN SUR MACINTOSH
par Goulven Habasque,
240 pages, Réf. 0099

ATARI

JEUX EN BASIC SUR ATARI *par Paul Bunn,*
96 pages, Réf. 282

ATARI, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 387

ATARI, GUIDE DE L'UTILISATEUR *par Thomas Blackadar,*
192 pages, Réf. 354

ATMOS

JEUX EN BASIC SUR ATMOS *par Pierre Monsaut,*
96 pages, Réf. 346

ATMOS, 56 PROGRAMMES *par Stanley R. Trost,*
180 pages, Réf. 372

COMMODORE 64

JEUX EN BASIC SUR COMMODORE 64
par Pierre Monsaut,
96 pages, Réf. 0017

COMMODORE 64, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 342

GUIDE DU BASIC VIC 20, COMMODORE 64
par Douglas Hergert,
240 pages, Réf. 312

COMMODORE 64, GUIDE DE L'UTILISATEUR
par J. Kascmer,
144 pages, Réf. 314

COMMODORE 64, 66 PROGRAMMES
par Stanley R. Trost,
192 pages, Réf. 319

COMMODORE 64, GUIDE DU GRAPHISME
par Charles Platt,
372 pages, Réf. 0053

COMMODORE 64, JEUX D'ACTION *par Eric Ravis,*
96 pages, Réf. 403

COMMODORE 64, 1ers CONTACTS
par Marty Dejonghe et Caroline Earhart,
208 pages, Réf. 390

COMMODORE 64, BASIC APPROFONDI
par Gary Lippman,
216 pages, Réf. 0100

DRAGON

JEUX EN BASIC SUR DRAGON *par Pierre Monsaut,*
96 pages, Réf. 324

EXL 100

EXL 100, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0126

GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL
par François Abella,
208 pages, Réf. 264

HECTOR

HECTOR JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 388

IBM

IBM PC EXERCICES EN BASIC *par Jean-Pierre Lamoitier,*
256 pages, Réf. 338

IBM PC GUIDE DE L'UTILISATEUR
par Joan Lasselle et Carol Ramsey,
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 359

GRAPHIQUES SUR IBM PC *par Nelson Ford,*
320 pages, Réf. 357

GUIDE DU PC DOS *par Richard A. King,*
240 pages, Réf. 0013

LASER

LASER JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 371

MO 5

MO 5 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0067

MO 5, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 370

MO 5, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 375

MO 5, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 384

MO 5, DYNAMIQUE CINÉMATIQUE, MÉTHODE POUR LA PROGRAMMATION DES JEUX *par Daniel Lebigre,*
272 pages, Réf. 0118

MSX

MSX, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 411

MSX, INITIATION AU BASIC *par Rodnay Zaks,*
248 pages, Réf. 410

MSX, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0109

MSX, GUIDE DU GRAPHISME *par Mike Shaw,*
192 pages, Réf. 0132

ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw,*
96 pages, Réf. 278

ORIC PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 344

SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2
par Michel Lhoir,
2 tomes, Réf. 261-262

SPECTRAVIDEO

SPECTRAVIDEO, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 377

SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM
par S.M. Gee,
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw,*
96 pages, Réf. 276

SPECTRUM, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 381

SPECTRUM JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 368

TI 99/4

PROGRAMMEZ VOS JEUX SUR TI 99/4
par François Abella,
160 pages, Réf. 303

TO 7

JEUX EN BASIC SUR TO 7 *par Pierre Monsaut,*
96 pages, Réf. 0026

TO 7, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 328

TO 7, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 350

JEUX SUR TO 7 et MO 5 *par Georges Fagot-Barraly,*
168 pages, Réf. 0134

GESTION DE FICHIERS SUR TO 7 ET MO 5
par Jean-Pierre Lhoir,
136 pages, Réf. 0127

TO 7, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 374

TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80
par Léopold Laurent,
2 tomes, Réf. 366-251

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsaut,*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer,*
96 pages, Réf. 302

JEUX EN BASIC SUR TRS-80 COULEUR
par Pierre Monsaut,
96 pages, Réf. 325

TRS-80 MODÈLE 100, GUIDE DE L'UTILISATEUR
par Orson Kellog,
112 pages, Réf. 300

TRS-80 COULEUR, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 414

TRS-80 COULEUR, 56 PROGRAMMES
par Stanley R. Trost,
160 pages, Réf. 413

VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20
par G. O. Hamann,
2 tomes, Réf. 329-337

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay,*
96 pages, Réf. 277

VIC 20, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 341

VIC 20 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 345

VG 5000

VG 5000, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 422

VG 5000, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0128

ZX 81

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert,*
208 pages, Réf. 351

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 304

GUIDE DU BASIC ZX 81 *par Douglas Hergert,*
204 pages, Réf. 285

JEUX EN BASIC SUR ZX 81 *par Mark Charlton,*
96 pages, Réf. 275

ZX 81 PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 343

MICROPROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks,*
618 pages, Réf. 358

APPLICATIONS DU Z80 *par James W. Coffron,*
304 pages, Réf. 274

PROGRAMMATION DU 6502 *par Rodnay Zaks,*
376 pages, Réf. 0031, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks,*
288 pages, Réf. 332

PROGRAMMATION DU 6800
par Daniel Jean David et Rodnay Zaks,
374 pages, Réf. 327

PROGRAMMATION DU 6809
par Rodnay Zaks et William Labiak,
392 pages, Réf. 0139

PROGRAMMATION DU 8086/8088
par James W. Coffron,
304 pages, Réf. 0016

MISE EN OEUVRE DU 68000 *par C. Vieillefond,*
352 pages, Réf. 0133

ASSEMBLEUR DU 8086/8088
par François Retoreau,
616 pages, Réf. 0093

SYSTÈMES D'EXPLOITATION

GUIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
354 pages, Réf. 336

CP/M APPROFONDI *par Alan R. Miller,*
380 pages, Réf. 334

INTRODUCTION AU p-SYSTEM UCSD
par Charles W. Grant et Jon Butah,
308 pages, Réf. 365

GUIDE DE MS-DOS *par Richard A. King,*
360 pages, Réf. 0117

APPLICATIONS ET LOGICIELS

INTRODUCTION AU TRAITEMENT DE TEXTE
par Hal Glatzer,
228 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman,*
200 pages, Réf. 0062

WORDSTAR APPLICATIONS *par Julie Anne Arca,*
320 pages, Réf. 0005

VISICALC APPLICATIONS *par Stanley R. Trost,*
304 pages, Réf. 258

VISICALC POUR L'ENTREPRISE *par Dominique Helle,*
304 pages, Réf. 309

INTRODUCTION A dBASE II *par Alan Simpson,*
280 pages, Réf. 0064

DE VISICALC A VISI ON *par Jacques Bourdeu,*
256 pages, Réf. 321

MULTIPLAN POUR L'ENTREPRISE
par D. Helle et G. Boussand,
304 pages, Réf. 0079

dBASE II APPLICATIONS *par Christophe Stehly,*
248 pages, Réf. 416

INTRODUCTION A LOTUS 1-2-3
par Chris Gilbert et Laurie Williams,
272 pages, Réf. 0106

LOGISTAT, ANALYSE STATISTIQUE DES DONNÉES
par Fredj Tekaia et Michele Bidel,
192 pages, Réf. 0132

La plupart de ces ouvrages existent en version anglaise.

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 42.03.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Vogelsanger. WEG 111
4000 Düsseldorf 30
Postfach N° 30.09.61
Tel. : (0211) 626441
Telex : 08588163

Paris • Berkeley • Düsseldorf

Achevé d'imprimer le 23 novembre 1985 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 4e trimestre 1985 - Imprimeur n° 1886

Ce livre contient toutes les informations nécessaires pour développer vos propres applications basées sur le Z80.
Il comporte de nombreux exemples et diagrammes concernant l'utilisation des éléments périphériques :

- ROM et RAM statique
- Circuits d'entrée-sortie
- RAM dynamique
- SIO
- PIO
- CTC

Un chapitre est consacré au traitement des interruptions et un autre aux communications série.

L'AUTEUR

JAMES W. COFFRON est titulaire d'un MSEE de l'Université de Santa Clara, il est ingénieur système spécialisé dans la conception et la mise au point des systèmes à microprocesseur. Il a enseigné la micro-informatique dans les milieux professionnel et universitaire. Il est l'auteur d'un grand nombre de livres sur les microprocesseurs.

0181 1185 210F00
9 782736 101817

SYBEX

JAMES W. COFFRON

APPLICATIONS DU Z80